More intelligent delivery of numerical analysis to a wider audience

Dupee, Brian J.

Award date:
1997

Awarding institution:
University of Bath

More Intelligent Delivery of Numerical Analysis to a Wider Audience

submitted by

Brian J Dupée

for the degree of PhD

of the

University of Bath

1997

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

To Elke

Acknowledgements

I would like to thank those that have helped and supported me during the course of preparing this thesis. In particular I would like to thank Prof. James Davenport who supervised the research and both Mike Dewar and Themos Tsikas at NAG Ltd.

I must also thank Dr. David Hornsby who was the first to believe that I could do it, but who foolishly bet that I would present my thesis before he.

Summary

Demand by users for modern easy-to-use graphical interfaces in the form of Windows and the World Wide Web browsers has led to the perfectly reasonable expectation that all computing should be done this way. In answer to this, many improvements, especially in terms of ease of use, have been made to software for solving mathematical problems, in particular symbolic packages such as Maple, Mathematica and Axiom, in that they now come complete with some Windows-type interface. However, many mathematical problems either cannot be solved using symbolic methods or such methods are unsuitable and numerical methods are preferable. The user has thus been forced to rely on older numerical software. This is proven software where separate numerical routines are generally collected together to form libraries, bringing into one package a considerable amount of technical and mathematical expertise.

The use of these libraries means that the user is forced to employ what can only be considered as archaic methods — writing and compiling programs in Fortran etc. Due to their historical development, such methods are difficult to use and difficult to understand when they should be used. This means that they are not suited to use by a novice or someone more used to Windows technology and there is little help in choosing the appropriate routine for any given problem, should there be a choice.

This thesis describes the construction using new technology of an intelligent interface to numerical library routines. In particular, it shows the techniques used in the construction of an expert system for choosing and applying NAG numerical routines in the fields of numerical integration, numerical solution of differential equations and optimization available to the Axiom computer algebra system. It also describes the construction of an easy to use windows interface using hypertext technology.

Contents

Part I

Background

Chapter 1

Introduction

A piece of software will not be used voluntarily unless it is easy to use, however good it may be internally.

[Kemp, 1978]

1.1 The Nightmare Scenario

An engineer has a problem. In some design process he has to estimate a minimum load on some linkages. He has done the mathematical modelling and goes to his supervisor to get help on performing the computation. The supervisor looks at the problem and tells the engineer that he should go and look at the NAG Numerical Library [NAG, 1996] since they're bound to have some routine for it.

The engineer goes to the shelf where the manuals are kept, and is faced with 12 large volumes, heavily bound, with titles which clearly say "D02", "E04" etc. He doesn't give up.

After a while he finds the 'Foreword' and reads that, for minimization, he requires the 'optimization' chapter E04 and pulls that volume off the shelf. He is then presented with a few hundred pages of technical jargon and Fortran 77 code. It starts to dawn on him that he's going to have to write a Fortran program. He doesn't give up.

After wading through the introduction he realises that the problem he has is a minimization subject to non-linear constraints and the decision tree

provided tells him to use E04UCF. He thinks he feels a little happier now that he knows he's on the right track and turns to the section on E04UCF, only to find that the section is 41 closely-typed pages of items labelled OBJFUN, ISTATE(N+NLIN+NCLIN) and CJAC(NROWJ,N) and that it's not just one program, but needs a number of sub-programs. He doesn't give up.

He realises that there is a sample program that he can use to try and understand how he can program his own problem. Using that, and an old Fortran 77 manual, he starts to type. All these variable names are so confusing, and all seem to be used quite arbitrarily. It's so easy to make mistakes. After three attempts to compile the example code, he breaks out in a cold sweat.

And wakes up!

This scenario goes some way towards explaining why programmers, and Fortran programmers in particular, are regarded as strange. They are increasingly thought of as the dinosaurs of the computing industry. They appear to live and breathe by the ethic that "if it's difficult to write, it should be difficult to understand!" But we're talking about thirty years of the life and work of brilliant men and women. If they could suffer by the rules of a programming language designed in the 50s and 60s which allowed Neil Armstrong to walk on the moon, don't we have a duty to offer our respect?

But computing is a different subject now. We are no longer in general constrained by the size of memory, or by a need to use punched cards for input. We can display the output of a program on a colour monitor whilst performing further calculations on a machine thousands of miles away. But still we cannot throw all this expertise away. We also don't wish to repeat all their work just to take advantage of more modern machines.

The answer is to get those more modern machines to do the work for us.

The same engineer has another problem. Buoyed up by his success in solving the minimization problem, he decides to tackle a problem where he is investigating the effect of a number of different loads on the structure over a period of time. He knows this is an ordinary differential equation and so is directed to the 'D02' chapter of the NAG library.

He surmises that his is an initial-value problem, but cannot get any further without knowing whether the problem is "stiff" or not. There is nothing

here that can tell him how to find out without doing some preliminary calculations using one of the routines and "conclusions based on the computer time used and the number of evaluations of the derivative function f_i". So not only does he have to write a Fortran program (and sub-programs) to perform the calculation, he has to write a Fortran program (and sub-programs) to work out which Fortran program he should write to perform the calculation.

This illustrates another difficulty with using numerical methods. The method itself is often tuned to a specific subset of problems and they are either incapable, or at least inefficient, when confronted with a problem outside this minor subset. So as well as requiring something which can make the use of Fortran routines a lot easier, there is a need for software which can at least give some help on the problem of choice.

This is an area which has been addressed in the past. "There are expert systems which attempt to analyse the problem as presented by the user and, possibly with user interaction, thence decide upon the best means of solution. This type of system is still in its infancy and it may be some years before complex numerical problems can be solved this way"[Hopkins & Phillips, 1988].

1.2 The Windows Revolution and the "New User"

Increasingly, users' introduction to computing and computing methods is by means of the PC running some Windows technology. Therefore many users' experience is with graphical interfaces which ensure that complicated programs are implemented in some logical and straightforward fashion. Undoubtedly this is how computers should be — *it is the user who commands, not the computer that demands.*

Until recently, all "serious" mathematical computing has been done on mainframes and their successors, the workstations, using a range of 'old' technologies i.e. Fortran programs. This has started to change, firstly with the introduction of improved user interfaces to UNIX programs (using, for example, X or its derivative, OpenWindows) and increasing use of web technology and hypertext in particular. Modern applications designers must start to take these technologies into account when constructing user interfaces for the newly computer literate.

Furthermore, applications builders should consider the premise that the user doesn't

need or want to know the inner workings of each individual method of computing an integral, for example. Such details should be left to the computer program and, whilst some explanation of which method has been used could be communicated back, the user only wishes that the computation be carried out and a sufficient answer reached.

1.3 The Proposal

The aim of this project is to show that such an expert system can be created using current technology. The actual implementation that will be attempted is to use numerical routines of the NAG Fortran Library from within, and using, the Axiom Computer Algebra System. This would then provide an intelligent interface to such numerical software. However, not all routines in the NAG library require much intelligence to decide on their use since any determination of attributes is fairly superficial, if any is necessary. For example, there are four routines in the NAG library to determine the roots of a polynomial: the user only needs to decide whether a given polynomial is quadratic or not and whether its coefficients are real or complex to select the appropriate one (although it is more likely that Axiom can efficiently perform such a task symbolically obviating the need for such numerical routines, although there are exceptions c.f. §6.2.1, p. 49). But there are chapters of the NAG library where there is considerable choice and, more importantly, where the criteria for that choice is less clear cut.

As indicated above, the chapters `E04` on Optimization and `D02` on Ordinary Differential Equations are prime candidates since the use of each requires considerable knowledge of numerical analysis as well as the time and patience to create considerable Fortran programs. A further chapter for consideration is `D01` on Integration, where the user must consider such difficulties as the continuity of the integrand or the presence of weight functions (c.f. §6.1.2, p. 46). If and when the developers of Axiom can provide links to more routines in chapter `D03` on Partial Differential Equations or when other numerical routines become available, it would be beneficial for its inclusion.

Whilst there is a need for a better interface to many of the other routines in the NAG library which are available to the Axiom system, their construction is not a suitable subject for this thesis. However, it is eminently reasonable to suggest that techniques and ideas used in this thesis, particularly the hypertext-style interface, could be implemented at little cost.

This thesis is in two sections. The first contains descriptions of the chosen problem domains and the technology. It begins by introducing the problems inherent in using numerical software and, in particular, the NAG Numerical Library routines. Previous attempts to provide a better and more intelligent way of using such library routines are discussed. These include systems which use a form of the decision tree provided by NAG in their documentation and early rule-based expert systems' efforts to provide automatic selection of numerical routines.

The rationale behind more modern interfaces to mathematical software is introduced using the examples of the Computer Algebra Systems Maple and Axiom, with particular reference to calculus and their use of numerical algorithms. This is followed by an introduction to expert systems and their construction.

The second section is a step by step guide to the proposed system. It begins with a discussion of the computational agents necessary to an expert analysis of the given problem and some of the techniques used to perform that analysis. For example, the algorithms required to answer such questions as *Is this function continuous?* and *How stiff is this set of differential equations?*

The next four chapters describe the different sections of the expert system. This includes a formal description of the construction of the knowledge base and a description of the theoretical basis for the decision process constructed by extending Dempster-Shafer theory and combining with Lucks/Gladwell intensity, compatibility and aggregation functions. Concluding this section on the expert system is a chapter covering why an explanation system is required and how such a mechanism is created.

This is followed by a description of the simplified user interface, both the command line structure and the hypertext style input process. The conclusion brings together all these separate areas and includes an evaluation of the implemented system.

Chapter 2

The Problems of Using Numerical Software

There are a number of ways to solve numerical problems in Mathematics or Engineering. Hitherto, one of the most reliable methods is to use code previously shown to be accurate and sufficiently efficient contained in one or more of the available Numerical Libraries. These house a number of programs or subroutines to perform either a complete calculation or part of the calculation.

However, it is not always a simple process:

- The interface to these routines has not substantially changed since the early 1970s.

  Whilst technological advances in computers and user interfaces has continued apace, the techniques generally associated with numerical computation has remained static. Where we see millions of new computer users a year, those able to use and understand these routines are dwindling.

- The problem has to be stated in a form similar to that which the library routine can use.

  Even though there may be different routines available for the solution of the problem, it is not always the case that they accept the problem statement in the same way.

- The language of the code of many of these routines may be unfamiliar to the user.

  Since it makes sense for the programming language to reflect the purpose and style of the originators of the code ([Du Croz, 1982]) and since the routines are

likely to have been written before the latest languages and design principles, it is quite possible for them to be difficult to understand. See [Dekker, 1980].

- It may not always be apparent which routine is best to solve a given numerical problem.
- Due to the diversity of its authors, the interfaces may be inconsistent.

  These inconsistencies may be as simple as parameters differently ordered e.g. the parameter `IFAIL` is often the last parameter in the list but occasionally elsewhere. It may be that two routines to perform the same task have totally different data parameters or the names for those parameters are different.

Many of the numerical libraries (NAG, IMSL, LINPACK etc.) use Fortran 77 as a programming language, and, however much experienced programmers regret the current state of affairs, Fortran is not a popular choice among students who would rather learn the latest programming fad. There do exist a number of libraries written in "more modern" languages, such as C or C++ but these tend to be smaller in scope and availability.

Whilst interoperability (calling Fortran subroutines from a C, C++ or other program) is almost certainly possible, the lack of standards engenders more difficulties than are likely to be envisaged [SunSoft, 1995, §12]. For example, array structures would almost certainly be different[1], the naming schemes for functions and variables are inconsistent[2] and many structures cannot be accommodated[3].

2.1 Using and Abusing Fortran

Fortran as a language was created in the late 1950s specifically for mathematical programming, and, since this was among the first uses of computers, still contains many

[1] In Fortran, array subscripts by default start at 1 as opposed to 0 in C, C++ etc. and the dimensions are ordered differently — Fortran stores arrays in *column major order* (usually in contiguous memory) and C (C++) in *row major order*. Transposing arrays is not trivial – it could be expensive as the obvious algorithm is cache-pessimal.

[2] Sometimes compilers/linkers require an additional underscore character to be added to external Fortran subprogram names (library procedures may require two underscore characters) and there is a variance with the treatment of upper and lower case characters in variables

[3] C and C++ allow a far greater range of data structures than is possible in Fortran although Fortran has the basic type `COMPLEX`, missing in most other languages. The `CHARACTER` type is so different in Fortran and C that its use is considered to be inadvisable.

reminders of its past[4]. It is difficult for today's programmers to understand the restrictions with which early implementations of numerical algorithms were created and used, particularly with respect to memory management and efficiency. It would be a steep learning curve indeed if a modern user was forced to come to grips with a computer language older than himself, when the hardware on which it is to be run is considered out of date in three years!

One of the major problems with Fortran is that since there may be many ways to code an algorithm, the demands of efficiency have led to obfuscated code. But fortunately the considerable testing process has vindicated each and every one so that we can almost use the routines as "black boxes".

However, the arcane naming structure causes great semantic difficulties, with its six character limit and implicit typing of variables. For example, the NAG Fortran Library Routine D01AJF, a routine for numerical quadrature which implements an adaptive scheme due to Piessens and De Doncker[5] (its use will be further discussed later in this thesis), has the following specification:

```
      SUBROUTINE //D01AJF// (F, A, B, EPSABS, EPSREL, RESULT,
     1                       ABSERR, W, LW, IW, LIW, IFAIL)
      INTEGER                LW, IW(LIW), LIW, IFAIL
      //real//               F, A, B, EPSABS, EPSREL, RESULT,
     1                       ABSERR, W(LW)
      EXTERNAL               F
```

In explanation, the user is required to provide:

[4]FORTRAN (FORmula TRANslator) was originally created for the IBM 704 as a replacement for machine and assembly languages which were the only way of instructing early computers. It held up the promise to be easy to use and understand, produce highly efficient machine code and virtually eliminate coding and debugging. In its day, it went some way to achieve this considering the incredible complexity of hand-written machine instructions [Backus, 1981]. The compiler (originally called the translator) was distributed to all users of the 704 computers in 1957. This was fairly quickly replaced by Fortran II, which added the concept of SUBROUTINE and FUNCTION.

Further development led to different version for each of a number of machines and the first attempt at standardisation in 1966. FORTRAN 66 thus became the preferred tool for mathematical programming, but was far from ideal. Due to the failure of the 1966 standardisation committee to produce a consistent and unambiguous document coupled with their concentration on performance issues instead of rationalisation, the new 'standard conforming' compilers allowed, and even encouraged, programs to be written in an increasing number of 'flavours' and 'dialects'. Whilst much of this was confronted in the first major revision in 1977, many of the 'horrors' (such as Hollerith formatting structures — created due to the failure of the 1966 document to institute proposals for a data structure for the character string) remained allowable. This can still be seen today in legacy code.

[5]This is itself base on Kronrod's version of Gauss-Legendre quadrature using a 10-point Gauss rule and a 21-point Kronrod rule.

F : An external //real//[6] Fortran function for the evaluation of the integrand at a given point

A : The lower limit of integration

B : The upper limit of integration

EPSABS : The required absolute accuracy

EPSREL : The required relative accuracy

RESULT : On exit, the approximation to the integral

ABSERR : On exit, an estimate of the absolute accuracy achieved

W(LW) : On exit, details of the computation

LW : The dimension of W

IW(LIW) : On exit, details of the computation

LIW : The dimension of IW

IFAIL : Failure warning characteristic

Whilst RESULT and ABSERR (absolute error) might be understandable to a modern programmer or mathematician, it is unlikely that any other requirement or parameter is obvious. Fortunately, apart from the name of the routine (D01AJF), it is possible with many modern compilers which have extensions to the language to use more meaningful names in the calling routine.

The problem exists that without knowing the data structures required by the individual routine and therefore some of the internal workings of the routine, it is difficult to use and comprehend. For this purpose, library suppliers produce large user manuals running into thousands of pages (the current NAG Fortran Library manual is a somewhat impenetrable 12 volumes). Much of this information is available on-line obviating the need for a weight-lifter's physique, but this does not help when the appropriate routine is not known. Furthermore, on-line manuals require different reading and navigation techniques or tend to become difficult to use and understand.

2.2 The Problem of Choice

For many types of problem, library suppliers produce a single routine to perform a single task. For example, to find the zeros of a complex polynomial, the NAG Fortran Library has the routine C02AFF. However, for some problem domains, there may be

[6] //real// indicates REAL or DOUBLE PRECISION depending on the implementation/machine architecture.

a choice of routine, as is the case for definite integration and numerical solution of ordinary differential equations. This choice of routine usually depends on attributes of the input problem, some of which might be easy to identify, others may be difficult or even impossible.

"Selecting the 'best' mathematical software requires a deep understanding of the problem domain and intimate familiarity with the available software. Since such combined expertise is rare, much currently available mathematical software is routinely misused." [Lucks & Gladwell, 1992, p. 12] There have been two traditional methods used by students and engineers of getting round this problem.

1. Work out the best method by studying the problem specification and the help sections of the user manuals. (This may be done by your local friendly Numerical Analyst if you can get him/her in the right mood.)
2. Find a routine that works (however inefficiently) by trial and error and continue to use that routine for all types of problems until the time comes when the the library supplier updates or withdraws the routine. Repeat.

As can be imagined, (1) above is probably better than (2) but the time involved could be considerable.

2.3 The NAG Fortran Library

NAG have been one of the foremost suppliers of library routines for numerical and statistical work for many years. The algorithms in the Fortran Library has undergone much research and improvement since the early 70s. They have become a well respected and widely available implementation of much of the best numerical code.

2.3.1 The Scope of the Library

In its latest incarnation, Mark 17 of the NAG Fortran Library [NAG, 1996] contains nearly 1200 routines, separated into 41 chapters or problem domains. Some of these chapters are of "utility routines", but the range of numerical routines is impressive. A major subset of the most used of 22 of these chapters has been incorporated within the Foundation Library for use on a wider set of platforms (see table 2.3.1)[7].

[7] On some platforms, due to the nature of any built-in functions or incorporated software, this list may be further reduced e.g where there exists built-in functions or software for performing, say, Linear Algebra, it would not make sense to provide duplication.

Chapter	Description	Routines in Full Library	Routines in Foundation Library
A02	Complex Arithmetic	3	0
C02	Zeros of Polynomials	4	2
C05	Roots of one or more transcendental functions	13	4
C06	Summation of Series	28	12
D01	Quadrature	29	12
D02	Ordinary Differential Equations	61	8
D03	Partial Differential Equations	24	3
D04	Numerical Differentiation	1	0
D05	Integral Equations	7	0
E01	Interpolation	15	10
E02	Curve and Surface Fitting	26	18
E04	Minimizing of Maximizing a Function	42	12
F01	Matrix Factorizations	24	10
F02	Eigenvalues and Eigenvectors	22	15
F03	Determinants	6	0
F04	Simultaneous Linear Equations	34	11
F05	Orthoganalization	1	0
F06	Linear Algebra Support Routines	173	0
F07	Linear Equations (LAPACK)	98	5
F08	Least Squares and Eigenvalue Problems	72	0
F11	Sparse Linear Algebra	10	0

Chapter	Description	Routines in Full Library	Routines in Foundation Library
G01	Simple Calculations and Statistical Data	48	19
G02	Correlation and Regression Analysis	56	10
G03	Multivariate Methods	18	3
G04	Analysis of Variance	7	0
G05	Random Number Generators	40	24
G07	Univariate Estimation	11	0
G08	Nonparametric Statistics	21	9
G10	Smoothing in Statistics	5	0
G11	Contingency Table Analysis	6	0
G12	Survival Analysis	2	0
G13	Time Series Analysis	38	16
H	Operations Research	6	0
M01	Sorting	17	6
P	Error Trapping	1	0
S	Special Functions	59	38
X01	Mathematical Constants	2	0
X02	Machine Constants	14	0
X03	Inner Products	2	0
X04	Input/Output Utilities	18	4
X05	Date and Time Utilities	4	4

Table 2.1: The NAG Fortran Library and Foundation Library

For example, there are 25 different top-level quadrature routines (11 in the Foundation Library)[8] from which the user must choose whichever is most appropriate to the problem in hand. Some of these may be for specific computer architectures, but most will depend on the attributes of the problem or the problem specification.

2.3.2 Programming using the NAG Library

When calling a library routine from an appropriate program there are a number of essential requirements [Hopkins & Phillips, 1988]:

- The **Name** of the Subroutine e.g. `D01AJF`
- **Data** parameters such as those which form part of the problem specification, e.g. the range of integration. Sometimes this data is in the form of an external Fortran function or subroutine to perform some evaluation or calculation.
- **Algorithmic Control** parameters such as error tolerances, iteration limits or how to deal with errors.
- **Housekeeping** parameters such as workspace or array dimensions.
- **Output** parameters (which may be combined with a data parameter as an **Input/Output** parameter) supplying the results of the calculation or any further information.

Many of these parameters will be of *basic types* i.e. Integer, Real (Double Precision) etc. or arrays of basic types. Some, however, may be Fortran subroutines (functions) in their own right – such as a function to evaluate an expression at some given point or a subroutine defining the Jacobian of a set of ODEs. These should be included in an `EXTERNAL` statement and if a function, the type explicitly declared.

Within the 12-volume user manual (and on-line from the NAG web site) there are for each routine example programs which give pointers towards their use. These examples are carefully chosen to highlight particular aspects of the routine and have the appropriate required attributes. The why's and wherefore's of such choices may not be explained and is therefore left to the user to determine whether a particular routine is or is not appropriate.

For example, the example program for the quadrature routine `D01AJF` is given below (`DOUBLE PRECISION` version). The NAG routines are identified as `EXTERNAL` (the value

[8] Not all routines within each chapter are top-level routines – some are utility routines or routines primarily called from within other routines.

of π is also given by an external function) and the expression for the integrand

$$I = \frac{x \sin(30x)}{\sqrt{1-(\frac{x}{2\pi})^2}}$$

is passed to the NAG routine as a Fortran function.

```
*     D01AJF Example Program Text
*     Mark 14 Revised.  NAG Copyright 1989.
*     .. Parameters ..
      INTEGER          LW, LIW
      PARAMETER        (LW=800,LIW=LW/4)
      INTEGER          NOUT
      PARAMETER        (NOUT=6)
*     .. Scalars in Common ..
      DOUBLE PRECISION PI
      INTEGER          KOUNT
*     .. Local Scalars ..
      DOUBLE PRECISION A, ABSERR, B, EPSABS, EPSREL, RESULT
      INTEGER          IFAIL
*     .. Local Arrays ..
      DOUBLE PRECISION W(LW)
      INTEGER          IW(LIW)
*     .. External Functions ..
      DOUBLE PRECISION FST, X01AAF
      EXTERNAL         FST, X01AAF
*     .. External Subroutines ..
      EXTERNAL         D01AJF
*     .. Common blocks ..
      COMMON           /TELNUM/PI, KOUNT
*     .. Executable Statements ..
      WRITE (NOUT,*) 'D01AJF Example Program Results'
      PI = X01AAF(PI)
      EPSABS = 0.0D0
      EPSREL = 1.0D-04
      A = 0.0D0
      B = 2.0D0*PI
      KOUNT = 0
      IFAIL = -1
*                      ..
      CALL D01AJF(FST,A,B,EPSABS,EPSREL,RESULT,ABSERR,W,LW,IW,LIW,IFAIL)
*
      WRITE (NOUT,*)
      WRITE (NOUT,99999) 'A      - lower limit of integration = ', A
      WRITE (NOUT,99999) 'B      - upper limit of integration = ', B
```

```
      WRITE (NOUT,99998) 'EPSABS - absolute accuracy requested = ',
     +  EPSABS
      WRITE (NOUT,99998) 'EPSREL - relative accuracy requested = ',
     +  EPSREL
      WRITE (NOUT,*)
      IF (IFAIL.NE.0) WRITE (NOUT,99996) 'IFAIL = ', IFAIL
      IF (IFAIL.LE.5) THEN
         WRITE (NOUT,99997) 'RESULT - approximation to the integral = ',
     +     RESULT
         WRITE (NOUT,99998) 'ABSERR - estimate of the absolute error = '
     +     , ABSERR
         WRITE (NOUT,99996) 'KOUNT  - number of function evaluations = '
     +     , KOUNT
         WRITE (NOUT,99996) 'IW(1)  - number of subintervals used = ',
     +     IW(1)
      END IF
      STOP
*
99999 FORMAT (1X,A,F10.4)
99998 FORMAT (1X,A,D9.2)
99997 FORMAT (1X,A,F9.5)
99996 FORMAT (1X,A,I4)
      END
*
      DOUBLE PRECISION FUNCTION FST(X)
*     .. Scalar Arguments ..
      DOUBLE PRECISION              X
*     .. Scalars in Common ..
      DOUBLE PRECISION              PI
      INTEGER                       KOUNT
*     .. Intrinsic Functions ..
      INTRINSIC                     SIN, SQRT
*     .. Common blocks ..
      COMMON                        /TELNUM/PI, KOUNT
*     .. Executable Statements ..
      KOUNT = KOUNT + 1
      FST = X*SIN(30.0D0*X)/SQRT(1.0D0-X**2/(4.0D0*PI**2))
      RETURN
      END
```

Whilst a good deal of the sample program is concerned with printing the output, it can be seen that the essentials are too easily hidden by detail and for the untrained or uninitiated it seems indecipherable. Even though this routine can perform the calcu-

lation, due to the oscillatory nature of the integrand, this is not the ideal routine for the given example problem. There is no indication to this effect in the documentation.

Instructions for calling the routines from C or C++ are equally obtuse, and there are a great number of inconsistencies [Hounham, 1996].

Chapter 3

Previous Attempts to Consider the Problem

3.1 Decision Trees – Help Numerical and GAMS

The NAG Numerical Library documentation contains a section which, in essence, is a decision tree for choosing the numerical routines. This supplements the detailed descriptions of each individual routine, but still can be intimidating for casual or inexperienced users. On-line versions of such documents further add to the difficulty since users are likely to find more problems navigating and are more difficult to scan.

The HELP Numerical package [Hazel & O'Donohoe, 1980] goes some way to providing the user with a keyword matching tree-based help system for the NAG and Harwell library routines. These and other library routines have been included in general decision tree programs e.g. NITPACK [Gaffney *et al.*, 1983], KASTLE [NAG, 1989] and the Guide to Available Mathematical Software (GAMS) [Boisvert, 1989] which is available on-line from the NAG web site.

Unfortunately, these systems suffer badly when choosing from a selection of routines which may all perform the calculation but with varying degrees of efficiency and accuracy. They are, basically, no more efficient in this respect than the documentation on which they are based. They also still require the user to answer some difficult questions about the attributes of the problem they require solved.

3.2 Rule Based Systems for Software Selection – NAXPERT, SAIVS and ODEXPERT

Much effort has been put into providing improved systems for choosing and using numerical library routines [Ford *et al.*, 1989]. Many of these use a knowledge base and a set of rules to assist in the decision process. Some also provide help in writing the program to call the routine [Schulze & Cryer, 1983].

These systems mainly use keyword matching and a set of rules based on the decision tree with perhaps a further knowledge base for help in constructing the appropriate Fortran code for calling the chosen routine. In this respect, NAXPERT, designed for a small mathematical library on IBM personal computers, has a set of about 50 Prolog rules along with a list of 160 keywords as a knowledge base. This is insufficient for the task since many consultations could not provide a recommendation or sufficient analysis.

Two systems, designed and built independently, together provide more insight into the requirements of the software selection process. These are SAIVS (Selection Adviser for Initial Value Software) [Lucks & Gladwell, 1992] and ODEXPERT [Kamel *et al.*, 1993] which both investigate the selection of numerical solvers of Initial Value problems of systems of Ordinary Differential Equations.

ODEXPERT uses an inference engine written in the rule-based programming language OPS 83 alongside computational processes to investigate the attributes of stiffness and structure (Fortran) and generation of the Jacobian matrix (Maple). The approach to investigating the problem of stiffness is particularly interesting. They implemented an algorithm to integrate the function backwards and compare the effect on the solution vector at each stage. The result of this test is then placed into one of the categories *stiff*, *mildly_stiff* or *non_stiff* since the inference rules could only work with qualitative attributes thereby losing much of the quantitative information it had gained.

The creators of the SAIVS system realised that any such inference engine must be able to use quantitative analysis of the attributes to get a deeper understanding of the problem and therefore instigate a more precise approach to software selection. In their view, ODEXPERT and NAXPERT "can be too inflexible, imprecise and qualitative for predicting software behaviour" [Lucks & Gladwell, 1992, p 12].

They thus based their system on the premises of feature intensity (e.g. *How acute is*

this attribute?), feature compatibility (e.g. *How does this affect the performance of the code?*) and evidence aggregation (e.g. *Does this interfere with or reinforce the effect of other attributes in the selection process?*). Thus attributes that interact competitively can be identified and their effects better incorporated into the selection process. This system will be further discussed in §6.2.1 and §8.2.

However, SAIVS does not include any computational agents for the automatic testing for these attributes, thus requiring the user to perform the appropriate analysis and enter the results during the interactive session. [Gladwell & Lucks, 1992]

3.3 Using a Rule Based System to Implement Library Routines – IRENA/ARC

A system was built which integrates a rule-based system for selecting quadrature codes with a link to the NAG subroutine library to form a seamless functional interface. The Automatic Routine Chooser (ARC) [Dewar, 1992] was written for the Interface between REduce and NAg (IRENA) [Dewar, 1991; Davenport *et al.*, 1991] and contained a set of LISP-like production rules and a number of computational agents for testing continuity, the presence of weight functions etc. This, together with the ability to create the Fortran stubs and call the appropriate routine automatically, simplified the interface by matching or pruning routines in the database.

For example, given a numerical integration to perform, ARC considers first the range of integration, choosing from its database routines suitable for either finite, semi-infinite or infinite ranges. It then considers the rules which can either match or prune routines from the list. By ordering all the routines remaining in the list after this process, an overall recommendation is made. Control then passes to IRENA which can automatically create any Fortran subprograms and call the library routines so solving the "having to write Fortran" problem.

The implementation only covered a subset of finite integration routines, but with the improved usability of the IRENA system provided a "black box" style using Reduce to perform the analysis, ARC to choose the routine which IRENA could call. The results are thus passed back to Reduce for display and dissemination.

The rule structure, as in ODEXPERT, was not appropriate for further development to other problem domains, especially where quantitative analysis was required, but

IRENA/ARC did point the way towards integrated software and was a significant precursor to the project described in this thesis.

Chapter 4

Computer Algebra Systems

4.1 Introduction

In the past thirty years, Computer Algebra Systems (CASs) have gone from the birth-pangs of initial research, through an intensive growing phase in the late 1960s and 1970s when basic algorithms for algebraic manipulation were refined and implemented, to today's maturing systems, adroit and masterly, each covering a wide area of mathematics and all with new and fairly straightforward (albeit very different) interfaces.

The basic tenet has remained, however, the same – to use symbolic and algebraic manipulation to find closed-form solutions to problems in a number of different domains. Various methodologies could be employed such as comparison with known forms, controlled simplification, substitutions etc. but in simple form it is the application of (mainly algebraic) computational rules on a mathematical object e.g. a polynomial or expression.

On the market today there are a number of CASs offering a wide variety of features – Maple, Mathematica, Macsyma, Derive, Reduce, Mupad and Axiom are the most common, all of which are available now on a number of computer systems. Some of these have extensions which allow them to use numerical methods as well as algebraic ones for the solution of certain problems such as definite integration and numerical solution of ordinary differential equations. I will describe some of the workings of Maple before concentrating on the CAS Axiom.

4.2 Maple

The newly released Maple V Release 4 is a fully-fledged interactive general purpose Computer Algebra System comprising a core program (short-term memory, manipulation tools, basic calculus tools etc.) and libraries of specialist code (compiled mainly from Maple programs) together with a relatively user-friendly interface. For a complete description see [Heck, 1996].

This interface handles the input and parsing of expressions, output of resulting expressions or function plots as well as displaying the help system, all in linked windows called worksheets. These worksheets can contain mathematical expressions, graphics or explanatory text with possibly hypertex-style links to other documents.

The commands are descriptions or abbreviations of descriptions of the functionality of a procedure. For example, `factor` and `Factor` factorise expressions over a given field, `int`, `Int`, or `integrate` are used to integrate a function (there are subtle differences to each form) and `dsolve` is used for the solution of Ordinary Differential Equations. The aim is to make the interface as intuitive as possible.

4.2.1 Integration

In operation, an integration problem is input as:

```
> f := 4/(x^2+1);
```

$$f := \frac{4}{x^2+1}$$

```
> int(f,x);
```

$$4\arctan(x)$$

Or, more readably as:

```
> Int(f,x): " = value(");
```

$$\int \frac{4}{x^2+1}\,dx = 4\arctan(x)$$

The definite integral is given by:

```
> Int(f,x=0..1): " = value(");
```

$$\int_0^1 \frac{4}{x^2+1}\,dx = \pi$$

For this particular integral, Maple uses look-up tables and substitution. For more difficult problems, it can call on firstly the Risch-Norman method [Davenport, 1982] or, if that fails, the Risch algorithm [Risch, 1969; Davenport, 1981]. However, for every integral that has a closed form solution, there are many that do not. The only way to calculate definite integrals of this form is to use numerical methods.

Maple has a number of strategies for numerical integration. The first is, again, using pattern-matching such as for the example:

```
> g := 1/(1+3*sin(t)^2):
> Int(g,t=0..2*Pi): " = value(");
```

$$\int_0^{2\pi} \frac{1}{1+3\sin(t)^2}\,dt = \pi$$

Whilst the answer is correct, this is, of course, a very dangerous tactic since the integral, which is

```
> Int(g,t): " = value(");
```

$$\int \frac{1}{1+3\sin(t)^2}\,dt = 2\frac{\arctan\left(2\frac{\tan(\frac{1}{2}t)}{4+2\sqrt{3}}\right)}{4+2\sqrt{3}} + \frac{\arctan\left(2\frac{\tan(\frac{1}{2}t)}{4+2\sqrt{3}}\right)\sqrt{3}}{4+2\sqrt{3}} - \frac{\arctan\left(2\frac{\tan(\frac{1}{2}t)}{4-2\sqrt{3}}\right)\sqrt{3}}{4-2\sqrt{3}} + 2\frac{\arctan\left(2\frac{\tan(\frac{1}{2}t)}{4-2\sqrt{3}}\right)}{4-2\sqrt{3}}$$

is discontinuous over the given range (see Figure 4.2.1).

```
> plot({g,int(g,t)},t=0..2*Pi, discont = true);
```

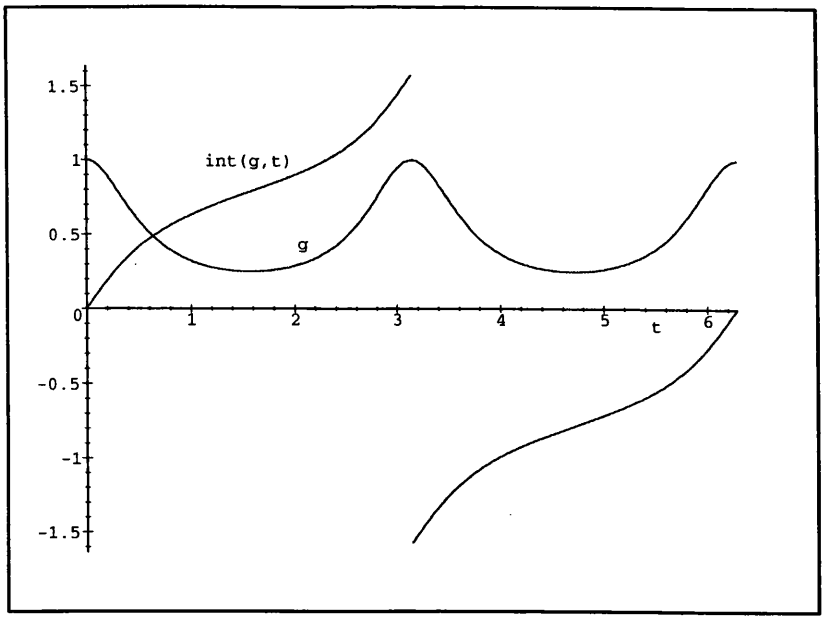


Figure 4.2.1: Plot of $\frac{1}{1+3\sin(t)^2}$ and its antiderivative.

It is much safer to use numerical methods:

```
> Int(g,t=0..2*Pi):  " = evalf(");
```

$$\int_0^{2\pi} \frac{1}{1+3\sin(t)^2}\,dt = 3.141592654$$

Since there is no discontinuity to the integrand (and even if there were, there are techniques to calculate an integral if one exists) it calculates the result using a Newton-Cotes method.

There are two other methods it can use for specific cases. For example:

```
> h := log(x*x):
> Int(h,x=-1..1):  " = value(");
```

$$\int_{-1}^{1} \ln(x^2)\,dx = -4 + 2\,I\,\pi$$

This result, calculated symbolically, while not wrong, is misleading since if such an integral exists, there exists a real answer[1]. Maple uses a double exponential method to to perform the calculation numerically:

```
> Int(h,x=-1..1):  " = evalf(");
```

[1]There are also fundamental questions of the existence of the integral and there is no indication that Maple has solved these.

$$\int_{-1}^{1} \ln(x^2)\,dx = -4.000000000$$

Any particular method can be forced on the integrator using an optional parameter. This will usually disable any singularity handling routine.

4.2.2 Ordinary Differential Equations

Maple has a number of tools for the solutions of ordinary differential equations i.e. equations of the form

$$F(y, y', y'', ..., y^{(n)}, x) = 0. \tag{4.1}$$

It can solve a limited range of these equations analytically. This depends very much on the type of equation i.e. ODEs of degree 1 and order ≤ 3 are sometimes possible. Outside of this range we are obliged to use numerical techniques.

If we take as our example the van der Pol equation ([Birkhoff & Rota, 1978, p. 134])

$$y'' - \mu(1 - y^2)y' + y = 0, \tag{4.2}$$

with $\mu = 1$ and initial values $y(0) = 2$, $y'(0) = 0$.

Let's first check that Maple cannot find an analytic solution:

```
> alias(y=y(t),y0=y(0),yp0=D(y)(0)):
> eqn := diff(y,[t$2])-(1-y^2)*diff(y,t)+y=0;
```

$$eqn := (\frac{\partial^2}{\partial t^2} y) - (1 - y^2)\,(\frac{\partial}{\partial t} y) + y = 0$$

```
> infolevel['dsolve']:=2:  # for feedback information
> dsolve(eqn,y);

dsolve/diffeq/linsubs:   trying linear substitution
dsolve/diffeq/missbody:   solving d.e. with missing variable
dsolve/diffeq/dsol1:   -> first order, first degree methods :
dsolve/diffeq/linsubs:   trying linear substitution
dsolve:   Warning: no solutions found
```

And now try the default numerical method:

```
> initvals := y0=2,yp0=0:
> F := dsolve({eqn,initvals},y,type=numeric);

dsolve/numeric:   entering
```

```
DEtools/convertsys:   converted to first-order system Y'(x) =
f(x,Y(x))      namely (with Y' represented by YP)
```

$$[YP_1 = Y_2,\ YP_2 = Y_2 - Y_2\,{Y_1}^2 - Y_1]$$

```
DEtools/convertsys:   correspondence between Y[i] names and original
functions:
```

$$[Y_1 = y,\ Y_2 = \frac{\partial}{\partial t}\,y]$$

```
dsolve/numeric:   vector Y of initial conditions at x0 =   0   array(1
.. 2,[(1)=2.,(2)=0])
```

$$F := \mathbf{proc}(rkf45\_x)\ \ldots\ \mathbf{end}$$

This has automatically changed the equation into a system of first-order odes and used a Feylberg 4th-5th order Runge-Kutta method. The output is a set of equations for t, y and y'. So, at $t = 10$:

```
> F(10);
```

$$[t = 10,\ y = -2.008340813688926,\ \frac{\partial}{\partial t}\,y = .03290706311963594]$$

From this, the function $y(t)$ can be plotted using:

```
> plot(t -> rhs(op(2,F(t))),0..20);
```

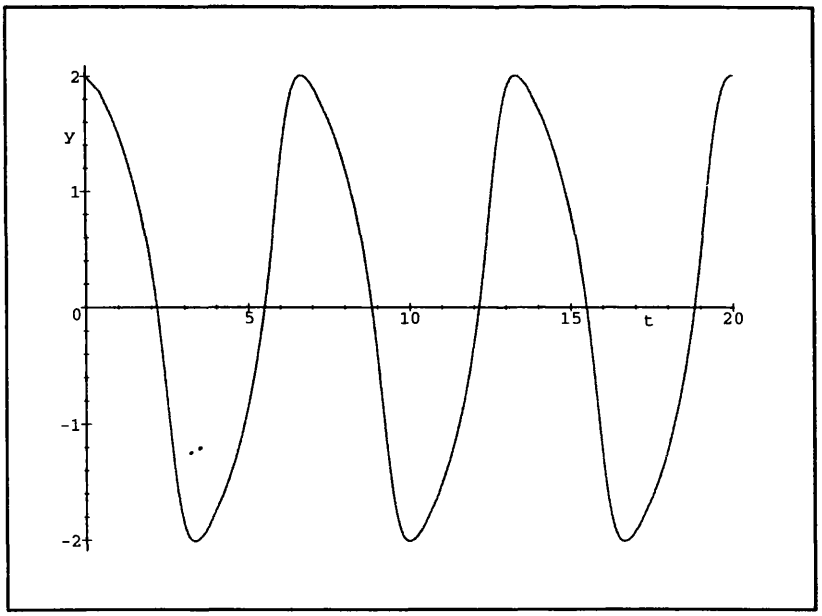


Figure 4.2.2: van der Pol's equation with $\mu = 1$, $y(0) = 2$ and $y'(0) = 0$.

The Maple ODE solver has the ability to use any of the 28 numerical solvers within the ODEPACK package. This includes both general and specific routines for stiff or non-stiff ODEs. The user has to specify the options `type = numeric, method =` *approach*, where *approach* is one of the specified routine names. Whilst the inclusion of such a large range of routines is admirable, the user is often left bewildered. However, the Livermore Stiff ODE solver (LSODE) is adaptive and can, as necessary, switch from the Backward Differential Formula (BDF) method for stiff equations to an Adams method for the non-stiff case [Hindmarsh, 1983].

4.3 Axiom

Without a doubt, compared to other CASs, Axiom is very different. This manifests itself in a variety of ways. Most apparent is its system of Types, Domains and Categories and its insistence that all things are 'Objects' and so have Types. For many users this seems to be a hurdle but, for the most part, this only clarifies what is taking place behind the scenes and can often be ignored. However, I will quickly describe the *raison d'être* of this system before introducing some of the built-in extensions to perform numerical computation.

4.3.1 The Object Oriented Paradigm

Most commentators in the field define a language as object oriented if and only if

- It supports objects that are data abstractions with an interface of named operations and a hidden local state
- Objects have an associated type [class]
- Types [classes] may inherit attributes from supertypes [superclasses].

[Cardelli & Wegner, 1985, p 481]

It can be seen that mathematical structures can fit in with this system. For example, an integer is an instance of the *class* of Integers ($\mathbb{Z}$); a polynomial with integer coefficients is an instance of the *class* of Polynomials over the Integers ($\mathbb{Z}[x]$). They each have *allowable operations* and, within the confines of the CAS, can maintain a hidden local state (i.e. it is not necessary for the user to know exactly *how* they are stored).

Each of the two 'objects' above belong to the Type *Ring* and share much of their allowable operations. These can thus be inherited from the Type. In this way, the complete basic algebra hierarchy can be accommodated. Furthermore, the object oriented paradigm was originally created to model data structures and complex elements [Booch, 1994], and as such are ideally suited to model structures of mathematical objects (lists, arrays, sets and tables).

4.3.2 Categories, Domains and Packages

Basic mathematical types are defined as Axiom[2] *Domains*, e.g. the integer 5 has type `Integer` and the polynomial x^2-2x+1 has type `Polynomial Integer`. However, types may not be unique – the integer 5 could equally well be defined as a `PositiveInteger`, `NonNegativeInteger` (both *Subdomains of* `Integer`) or even `IntegerMod(7)` (in which cases the allowable operations are altered since members of the Ring of Integers Modulo 7 ($\mathbb{Z}_7$) have multiplicative inverses, and Positive Integers ($\mathbb{Z}^+$) do not have additive inverses).

The allowable operations for each domain are either inherited from its *class* or Category as associated with its *attributes* or they are explicit to the specific domain or subdomain.

The Domains themselves belong to *Categories*. For example, the category `Ring` designates the class of all rings. This structure ensures the correctness of type. We can thus define functions which operate on, say, `Matrix(R: Ring)` without needing to specify *which* ring.

So, Domains can be Algebraic ones (like Integer, Polynomial or Matrix) or data structures (like lists or tables). One can build further types from these such as matrices of polynomials, or lists of integers or even lists of matrices of polynomials. However, polynomials with coeffients of type list are not possible.

It is the Category structure which ensures that the types are mathematically correct and allow functions to be created which operate on arbitrary types. These can be collected together within Packages and compiled thus extending the capabilities of the system. Indeed, most of the Axiom system has been created using Packages containing code for the creation and use of Categories and Domains.

[2]For a full description of the technical workings of Axiom, see [Jenks & Sutor, 1992].

4.3.3 Integration

In many ways, the interface to Maple and Axiom are similar. Each attempt to maintain a consistency of expression concordant with mathematical thinking and practice. So input of the integration problems in §4.2.1 in Axiom are[3]:

```
(1) -> integrate(f := 4/(x^2+1),x)

   (1)  4atan(x)
                                        Type: Union(Expression Integer,...)
(2) -> integrate(f,x=0..1)

   (2)  %pi
                Type: Union(f1: OrderedCompletion Expression Integer,...)
```

It can be seen that instead of **int** we now use **integrate** and along with the answer we are presented with the domain name. The second example is treated very differently to Maple:

```
(3) -> integrate(g := 1/(1.0+3*sin(t)^2), t=0..2*%pi)

   (3)  potentialPole
                                    Type: Union(pole: potentialPole,...)
```

If we force it to evaluate as if the pole doesn't exist we actually get an incorrect answer:

```
(4) -> integrate(g, t=0..2*%pi, "noPole")

   (4)  0
                Type: Union(f1: OrderedCompletion Expression Integer,...)
```

However, Axiom is supplied with a link to the NAG subroutine library [Dupée & Davenport, 1996; Broughan *et al.*, 1991; Hawkes & Keady, 1995] allowing the user to evaluate the integral numerically. The interface for this is more difficult but successful:

```
(5) -> d01ajf(0.0 ,%pi*2 ,0.0 ,1.0e-4 ,800 ,200 ,-1 ,g :: ASP1(F))

   (5)
   [w: Matrix(DoubleFloat), abserr: DoubleFloat, iw: Matrix(Integer),
    result: DoubleFloat, ifail: Integer]
                                                          Type: Result
(6) -> %.result

   (6)  3.14159265372921
                                                       Type: DoubleFloat
```

[3]It is unfortunate that the current output style for Axiom using UNIX machines does not have the flexibility of Maple.

4.3.4 Ordinary Differential Equations

The Package `ElementaryFunctionODESolver` provides an operation `solve` for finding closed form solutions of equations of type 4.1. Since everything must have a type, performing the operation is slightly different than using Maple. Taking example 4.2 we input:

```
(1) -> y := operator 'y

   (1)  y
                                                        Type: BasicOperator
(2) -> deq := D(y(x),x,2) - (1-y(x)^2)*D(y(x),x) + y(x) = 0
          ,,              2       ,
   (2)  y  (x) + (y(x)  - 1)y (x) + y(x)= 0

                                        Type: Equation Expression Integer
(3) -> solve(deq,y,x)

   >> Error detected within library code:
   getfreelincoeff: not a linear ordinary differential equation
```

This was, of course, a little optimistic [Postel & Zimmerman, 1996]. Numerical methods, using the NAG library, have a completely different interface:

```
(4) -> d02bbf(20.0, 200, 2, 0, 0.0, [[2 , 0 ]],0.0001, -1,([Y[2] , Y[2]-
Y[2]*Y[1]^2-Y[1] ]::Vector Expression Float)::ASP7('FCN),([i/10 for i
in 1..200]::Vector MachineFloat)::ASP8('OUTPUT))

   (4)
   [ifail: Integer, tol: DoubleFloat, result: Matrix(DoubleFloat),
    y: Matrix(DoubleFloat), x: DoubleFloat]
                                                          Type: Result
```

This calls the Runge-Kutta routine (`d02bbf`) directly with the system of first-order differential equations, producing output at 200 points between 0.0 and 20.0. These can then be plotted (see Fig 4.3.4):

```
(5) -> draw([i/10.0 for i in 0..200],column(%.result,1))
   Graph data being transmitted to the viewport manager...
   AXIOM2D data being transmitted to the viewport manager...

   (5)  TwoDimensionalViewport: "AXIOM2D"
                                          Type: TwoDimensionalViewport
```

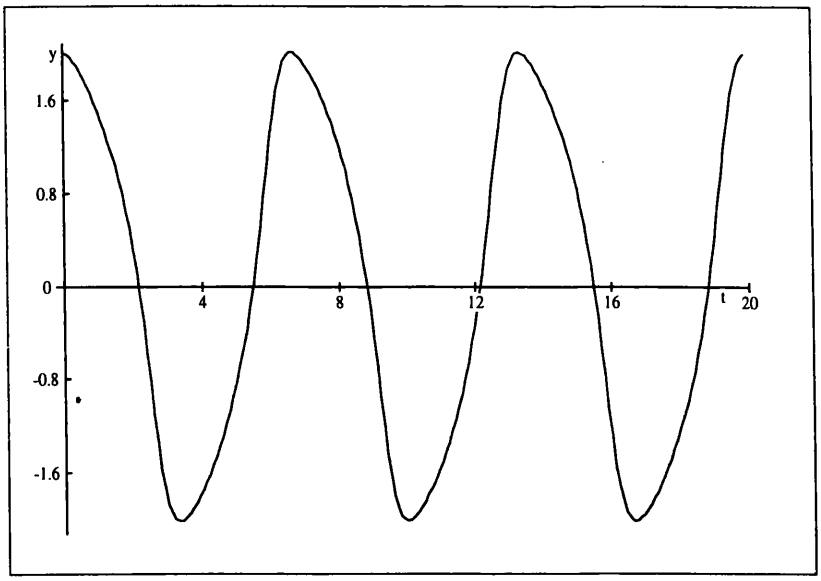


Figure 4.3.4: van der Pol's equation with $\mu = 1$, $y(0) = 2$ and $y'(0) = 0$.

4.4 Conclusion

Computer Algebra Systems like Maple and Axiom can indeed perform a lot of calculations for us. They can also call upon the power of numerical methods where closed form solutions either do not exist or are difficult to compute. However, there are drawbacks. The interfaces to these routines are not intuitive and whereas both systems provide help, much of this help is obtuse or uninformative to those "not in the know".

Given that these methods exist, there must be a better way to implement them to give the user much more valuable information and provide an interface more consistent with the remainder of the CAS.

Chapter 5

Expert Systems

5.1 Introduction

The essential characteristics of an expert system are normally regarded as[1]:

- containing a *knowledge base* i.e. some knowledge of a problem domain[2] contained in a distinct and identifiable form representing human or expert learning about that domain. This could be anything from a simple database, to a set of interacting systems.
- having the ability to perform some *reasoning* on this knowledge (typically simulating human reasoning) in addition to performing any mathematical calculation or modelling. This would normally use *heuristic* methods which do not guarantee success but exploit rule-of-thumb techniques to achieve *propositions* with varying degrees of certainty.
- having mechanisms to explain its recommendations and justify its reasoning sufficiently to convince the user of its correctness.

How these are organised and used varies greatly with the individual requirements. In

[1]It is very difficult to find some agreement or consensus amongst experts on the definition of an expert system. Some relax the requirement for an explanation mechanism, others insist that the knowledge acquisition phase is an integral part of the system. Some commentators define expert systems only in terms of other expert systems. [Frost, 1986] defines an expert system as:

> ... a system which is capable of carrying out a task generally regarded as being difficult and requiring some degree of human expertise.

[2]It is apparent that there could be some confusion of terminology between problem or knowledge *domain* and Axiom Domains. As far as possible I will use capitalisation on Axiom Domains only.

some expert systems in the past, various techniques for setting goals and subgoals, rule-sets and other control features have been implemented in varying quantities and combinations to tailor the particular expert system to the task.

If we consider the Expert System's knowledge base and inference mechanism as a form of Information Processing System, as defined by the Theory of Human Problem Solving ([Newell & Simon, 1972]), we can look, analogously, at the requirements as part of a problem space which consists of:

1. A *set of elements*, U, which are symbol structures, each representing a state of knowledge about the task.

2. A *set of operators*, Q, which are information processes, each producing new states of knowledge.

3. An *initial state of knowledge*, u_0, which is the knowledge about the task that the problem solver has at the start of problem solving.

4. A *problem*, which is posed by specifying a set of final desired states, G, to be reached by applying operators from Q.

5. The *total knowledge available* to a problem solver when he is in a given knowledge state, which includes (ordered from most transient to most stable):

   (a) *Temporary dynamic information* created and used exclusively within a single knowledge state.

   (b) The *knowledge state* itself — the dynamic information about the task.

   (c) *Access information* to the additional symbol structure held in *Long Term Memory* (LTM) or *External Memory* (EM) (the *extended knowledge state*).

   (d) *Path information* about how a given knowledge state was arrived at and what other actions were taken in this state if it has already been visited on prior occasions.

   (e) *Access information to other knowledge states* that have been reached previously and are now held in LTM or EM.

   (f) *Reference information* that is constant over the course of problem solving, available in LTM or EM.

[Newell & Simon, 1972, p. 810]

This has led to a representation of an ES which is essentially concentrated in providing a large, uniform description of knowledge and a relatively simple inference mechanism. There are considerable benefits to such a picture although, by necessity, due to the diverse nature of some ESs, this is not always achievable or desirable.

5.2 Examples of Expert Systems and Expert System Languages

In many respects, early expert systems have been task-driven, in that there is an easily identifiable goal and a set of rules to prove that goal or to prove subgoals leading to the goal.

If this is to identify the molecular structure of a chemical compound, as in CONGEN [Carhart, 1979] based on the Stanford University project DENDRAL, the goal is to identify the structure completely. The rules are constraints, either necessary or forbidden, so that, given part of the chemical, the next element must either belong to, or not belong to, some particular subset of elements. The heuristic is to "test and discard" and the knowledge base is restricted to the constraints and the rules on how to change these constraints.

The knowledge base for the 1972 expert system for blood infections, MYCIN [Buchanen & Shortliffe, 1984], was far more extensive in that it contained rules of the form:

> if *condition* 1 (and *condition* 2 ... (and *condition* m)) then
> assert *conclusion* 1 (and *conclusion* 2 ... (and *conclusion* n))

as well as a database of organisms and drugs. The control structures were more complex allowing subgoals to be easily verified and leading to a system of *backward chaining*. This is reasoning back from what it wants to prove towards the conditions that it needs to satisfy.

These techniques have been extensively used in expert system shells (prototype expert systems stripped of their domain knowledge), notably the MYCIN derivative EMYCIN (Empty MYCIN) containing the rule-based language, an organisational structure for the rules, the backward-chaining control, an interface to both create and edit the rules and user-interface. EMYCIN was used to great effect in expert systems such as PUFF investigating pulmonary function data.

The alternative rule structure, *forward chaining*, used in the data-driven expert system

created for VAX computer component arrangement, R1/XCON [Jackson, 1992, §17.2], written in the expert system shell OPS4 and re-written in OPS5, requires a different control mechanism to resolve conflicts where it might happen that more than one rule may be able to fire at a particular time. Apart from these 3 (or 4) conflict resolution rules, there is no control over the firing of rules. Instead of looking at the question *Since we wish to prove A, what do we need to show?*, it considers *Given that we know A, B, ... , what can we assume?, and will that get us nearer our goal?*

Many expert system shells have the feature that much of the algorithmic control normally contained in computer programs is withdrawn. For many applications this does not pose any problem (except in gaining a complete understanding of all the processes when used in a large system) but it does have major problems when extensive explanation and recovery procedures are required.

Expert systems can also be written in so-called "logic" languages i.e. Lisp (sometimes categorised as a "functional" language) or Prolog using all the necessary control structures and algorithmic complexity as required. In recent years, many logic, functional and object-oriented languages have been used to create expert systems, notably Lisp, Prolog and LOOPS.

5.3 The Knowledge Base

Knowledge can be divided into two forms — *Facts* i.e. atomic assertions that certain information is true, and *Rules* or *Heuristics* i.e. assertions that given a certain fact or facts, what other facts are therefore true, or can be assumed true. For example, given the Prolog predicates:

```
parent( fred, john). % Fred is a parent of John
parent( john, susan). % John is a parent of Susan
grandparent( X, Z) :- parent( X, Y), parent( Y, Z).
sibling( Y, Z) :- parent( X, Y), parent( X, Z).
```

the first two are facts and the third is a rule such that we can infer that Fred is a grandparent of Susan, but cannot learn anything from the `sibling` predicate. If we later learn that:

```
parent( john, alan).
```

we can immediately (if we so wanted) assert that Fred is also the grandparent of Alan and also that Alan is a sibling of Susan.

This organisational paradigm does not preclude that the 'fact' might be that we can place a given degree of certainty to a particular piece of information. We can therefore build *judgement* into the system. Given the variety of expert systems and knowledge domains, the representation of knowledge described above is just one of many possibilities. However, the essential feature is that it embodies the knowledge of an expert within the domain.

Alternative organisations for this knowledge exist. One of these is the *Frame* whereby a collection of 'slots' or separate pieces of information associated with a distinct entity are brought together as a single 'symbol structure'. When such information is filled i.e. complete, the frame is thus said to be *instantiated.*

Obtaining this knowledge can be achieved by:

- Extraction from written sources e.g. textbooks, reports, case-studies etc.
- Interview of the domain expert(s), possibly over considerable time.
- Induction from examples

Where the issues are fairly well documented, if complicated, written sources contain a valuable supply of domain knowledge, especially if a consensus is required. Domain experts would, theoretically, be excellent sources of information but obtaining that knowledge in usable form is particularly difficult. Examples can also be a useful additional source of information.

5.4 The Inference Machine

The main task of the inference machine is to use the knowledge contained within the knowledge base, together with any other knowledge it can elicit, to achieve a specified goal. This may include exploiting links between certain types of data as defined by any predefined rules or strategies or asking pertinent questions of the user or external agents.

It is quite possible that this is the smallest part of the expert system but mistakes here can have disastrous consequences and can be the most difficult to trace and correct.

It is, however, inextricably linked to the explanation process, since the logic of the

system and its inference mechanism must be imparted to the user in such a form that the process is understandable.

5.5 Explanation Mechanisms

The credibility of any expert system is likely to be dependent on its ability to justify and explain its reasoning. Since the task allotted to an expert system, or an expert, is more likely than not complex both in its demonstration and in its description, the user, whether another expert or a novice, must be able to follow the inference steps. Many expert systems founder because either too much jargon is used or much of the explanation concerns information already known.

[Weiner, 1979] identifies a number of important features including:

- Explanation should be limited to what is not already known to the user. It should therefore not be a restatement of the initial problem or its constraints.
- Details should not be given initially. It must be up to the user to *ask* for explanations otherwise the system will be seen as tedious and contrived.
- Details should be given in increments. There should thus be a hierarchy of explanation whereby the most technical of details are given only on the express command of the user.
- Explanations should be 'marked' in some way so that the underlying structure is more transparent. Thus explanation emanating from different parts of the inference process should be separated and identified.

5.6 Language Choice for the Proposed System

Given that the domain of the proposed expert system is Numerical Analysis, Prolog is unsuitable without extensive additions since it does not have the richness of mathematical constructs. Lisp itself would require large amounts of library code before it would be sufficient. So a low level language brings with it considerable development time.

A possibility would be the use of an expert system shell. Since our proposed expert system needs greater control of path-lines and would require many of these extra controls

to be explicitly reinstated, the use of a shell such as OPS5 [Brownston *et al.*, 1985], although written in Lisp and could easily access the Axiom system, would entail such re-writing as to be impractical.

As a symbolic language containing many characteristics reminiscent of Lisp, on which it is based, the Axiom language has some obvious advantages when we consider the domain of the tasks that we expect of it. It has been constructed with mathematical concepts, structures and operations in mind. Since it has a full panoply of control structures, we can tailor these to the task in hand, using as much, or as little, as necessary.

Whilst inference rules must be explicitly written in Axiom, because of its extensive library structure, inquiry and manipulation of required information is more logical, given the problem domain. The inclusion of computational agents, constructed using Axiom, for elicitation of further information would then be much more natural than using special inter-language constructs.

Part II

ANNA

Chapter 6

Computational Agents

> *It is a matter of perfect indifference where a thing originated; the only question is: "Is it true in and for itself?"*
>
> *G. W. F. Hegel (1770 - 1835)* Philosophy of History

Computational Agents are programs which can be called by an expert system to answer specific questions about the current state. This may be just a simple look-up or it may be much more complicated, involving considerable calculation, evaluation or interpretation. However involved this may be, within the context of an expert system we must always keep in mind that efficiency is paramount.

6.1 Integration

There are a number of criteria affecting the choice of routine for numerical integration:

- Is the function continuous?
- How oscillatory is it?
- Is the range finite?
- Is there a weight? i.e. Is the integrand factorisable such that one of the factors is of a specific form? See §6.1.2.

I have therefore created computational agents to provide answers to these and other questions [Dupée & Davenport, 1995].

6.1.1 Testing for Continuity

One of the major differences between algorithms for numerical integration is in their treatment of singularities. The question of "whether this or that function is continuous" is fundamental to many areas of mathematical analysis. Techniques have grown up through the years to address the problem. The most common method in mathematics, that of the $\delta - \epsilon$ argument, underpins much of analysis today. But abstract arguments such as this cannot, yet, be achieved by Computer Algebra systems[1]. So how can we decide whether a function is continuous and, if not, where do its singularities lie? Or where *might* they lie? Added to this we would have to add the question: where might there be a problem which, even though it could be continuous, might (due to the nature of computers in general, the particular platform used or numerical algorithms *per se*) cause a computational error?

In general, some of these problems have been shown to be undecidable[2] [Richardson, 1968] and whilst various techniques show promise (Padé approximation etc.), computation time has yet to be considered. As such, a complete answer to these questions is not the goal — the requirement is only for a workable algorithm for finding possible singularities to a function.[3]

The functions that will be used, i.e. **Elementary Real Functions**, are those real valued expressions of a single variable which can be defined relatively easily using a finite number of polynomial, logarithmic, exponential, and trigonometric operations [Geddes *et al.*, 1992, p. 512]. There are two types of test for continuity required. The first is allied to the search for end-point singularities of the algebraico-logarithmic type i.e. of the form

$$f(x) = (x-a)^c(b-x)^d \log(x-a)\log(b-x)g(x) \quad | \quad x \in [a,b]. \qquad (6.1)$$

[1] Whilst theorem provers, today still at a fairly simple experimental level, *might* be able at some stage to mirror these arguments successfully, their use is not an option, given the nature of the problem i.e. we are looking for *computational* continuity or more particularly we are trying to identify where a Fortran program could fail. For example, the function $f(x) = \frac{\sin(x)}{x}$ is continuous at $x = 0$. But if a Fortran program attempted to evaluate f at 0, it will undoubtedly fail.

[2] As in Theorem 3 of [Richardson, 1968], we multiply a discontinuous function by a function whose identity to zero is undecidable to prove that continuity is undecidable.

[3] Since these tests will be applied in an expert system which chooses numerical routines for the solution of various problems, and that these routines require, in the main, Fortran subroutines for the evaluation of the function, indeterminate forms, such as 0/0 need to be flagged as singular points even though l'Hôpital's rule might show that there exist non-singular ways of evaluating this expression. This is necessary since, in Fortran, expressions of the form 0/0 are undefined and therefore will, depending on the platform, signal an error.

This could be satisfied by use of power series methods since it is sufficient that if a power series limit does not exist at an end point then a singularity exists. The identification of the values for c and d in equation (6.1) is considered in §6.1.2.

So let us look at the code to test for continuity of an expression at a given point.

```
continuousAtPoint?(f:Expression Fraction Integer, e:Equation OrderedCompletion
                                        Expression Fraction Integer):Boolean ==

  l := limit(f,e)$PowerSeriesLimitPackage(Fraction Integer, Expression Fraction Integer)

    -- if the left hand limit equals the right hand limit, or if
    -- neither limit can be found, the return type of limit(f,e) is
    -- Union(OrderedCompletion Expression Fraction Integer,"failed")

  l case OrderedCompletion Expression Fraction Integer =>
    finite?(coerce(l)@OrderedCompletion Expression Fraction Integer)
  false
```

This fulfills all the requirements of the definition of continuity at a point[4]. It handles continuity at infinity[5] by substitution of the variable to bring it to a finite point. It even deals with the problems that occur when the function is not infinitely differentiable since if a Puiseux[6] series cannot be found, it uses exponential expansion[7] to find the limit [Knuth, 1981].

The second need for a test of continuity is a general search for all singularities within (interior to) the range of integration. A workable strategy for dealing with functions with multiple singularities is to split the function at those points and integrate over each segment separately. In this case we need to know not only that such singularities exist, but also where in the range of definition they are.

Given that we know when certain operations give rise to singularities, it is possible to search for these within an expression. This technique relies on the pattern matching

[4] A function f is said to be continuous at point c if and only if $\lim_{x \to c+} f(x) = \lim_{x \to c-} f(x) = f(c)$

[5] The Axiom type `OrderedCompletion Expression Fraction Integer` is the same as `Expression Fraction Integer` with the points `%plusInfinity` and `%minusInfinity` added.

[6] Variations on the Taylor series are the **Laurent series** which can have a finite number of terms of *negative* degree, and the **Puiseux series** having terms of *fractional* degree [Davenport *et al.*, 1988].
Even at a singularity, many functions have a valid Laurent or Puiseux series without, of course, a valid Taylor series. So a useful test for continuity at a particular point is to create a power series and test the leading exponent. If it is negative, the function contains a singularity at that point.

[7] The package writes the expression in the form of a quotient of exponential sums, each sum being a Puiseux series multiplied by an exponential of a Puiseux series. Lazy evaluation allows the limits to be calculated.

ability of algebra packages and on our knowledge of functions to give us a complete list of singular or problem points.

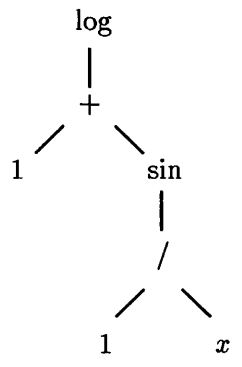
The first of the techniques involved searches for particular operations in the expression which could give a singularity. For example, we know that for $x = 0$, the value of the expression $\frac{1}{x}$ is not defined. So anything within the denominator of an expression which could be zero should be identified.

This would need to be handled recursively considering the range of definition of the input variable. The expression is looked at in terms of its expression tree. It is best to consider an example.

Example 6.1.1 *The function*

$$f(x) = \log\left(1 + \sin\left(\frac{1}{x}\right)\right)$$

has many singularities between 0 *and* 1.



The first part to consider is the expression $g(x) = \frac{1}{x}$. *This has a singularity at* $x = 0$ *and* $g(x)$ *has a range* $[1, \infty)$.

The expression $h(g(x)) = 1 + sin(g(x))$ *will have zeros at* $g(x) = \frac{(4n-1)}{2}\pi$ *but no singularities, but* $f(x) = \log(h(x))$ *will have singularities at all of these zeros.*

So the singularities of the function $f(x)$ *are at the points* $[0, \frac{2}{3\pi}, \frac{2}{7\pi}, \frac{2}{11\pi}, \cdots]$.

Thus there is a need for an algorithm which will look for operations for which singularities can occur, and search for possible causes of these singularities. Such pattern matching may already been done by the input parser and we only need access to the parser output. This is not the case with Axiom and we have to create explicitly an expression tree.

We can institute a look-up table containing the different elementary functions and any points at which they remain undefined together with some simplifying rules for expressions with singularities.

Let a, b be elementary functions and ρ, σ be functions such that ρ = SingularitiesOf

and σ = ZerosOf.

$$\begin{aligned}
\rho(ab) &\subseteq \rho(a) \cup \rho(b) \\
\rho(\frac{1}{a}) &= \sigma(a) \\
\rho(e^a) &= \rho(a) \\
\rho(a+b) &\subseteq \rho(a) \cup \rho(b) \\
\rho(\log a) &= \sigma(a) \\
\sigma(ab) &\subseteq \sigma(a) \cup \sigma(b) \cup \rho(ab) \\
\sigma(\frac{1}{a}) &= \rho(a)
\end{aligned}$$

These form a set of production rules which can be used in the search for singularities as well as those of more specific nature such as trigonometric expressions for which the look-up table is used (see tables 6.1 & 6.2). Since we are being conservative, we can replace the $\subseteq$ by $=$ above. This may overestimate the number of singularities, including the possibility of indeterminate forms, but such points can be investigated using series methods should the need arise.

Operation $f(x)$	Singularities at $x =$	Other Information
$\frac{1}{(x-a)}$	a	
$\log(x)$	0	Undefined on $(-\infty, 0)$
$\tan(x)$	$\frac{(2n+1)\pi}{2} \mid n \in \mathbb{Z}$	
$\sec(x)$	$\frac{(2n+1)\pi}{2} \mid n \in \mathbb{Z}$	
$\csc(x)$	$n\pi \mid n \in \mathbb{Z}$	
$\cot(x)$	$n\pi \mid n \in \mathbb{Z}$	

Table 6.1: **Part of a Look-up Table for Singularities**

Once a function is found which can have singularities, we need to evaluate both whether it is in the given range and which points in the range. Since the input range for each function may not necessarily be the same as the range of x, such points must be evaluated using inverse functions. These inverse functions can be evaluated recursively.

Operation f	$f(x) = 0$ at $x =$	$f(x) = 1$ at $x =$	Other Information
$\log(x)$	1	e	Undefined on $(-\infty, 0]$
$\sin(x)$	$n\pi \mid n \in \mathbb{Z}$	$\frac{(4n+1)\pi}{2} \mid n \in \mathbb{Z}$	
$\cos(x)$	$\frac{(2n+1)\pi}{2} \mid n \in \mathbb{Z}$	$2n\pi \mid n \in \mathbb{Z}$	
$\tan(x)$	$n\pi \mid n \in \mathbb{Z}$	$\frac{(4n+1)\pi}{4} \mid n \in \mathbb{Z}$	$x \neq \frac{(2n+1)\pi}{2}$
$\sec(x)$	—	$2n\pi \mid n \in \mathbb{Z}$	$x \neq \frac{(2n+1)\pi}{2}$
$\csc(x)$	—	$\frac{(4n+1)\pi}{2} \mid n \in \mathbb{Z}$	$x \neq n\pi$
$\cot(x)$	$\frac{(2n+1)\pi}{2} \mid n \in \mathbb{Z}$	$\frac{(4n+1)\pi}{4} \mid n \in \mathbb{Z}$	$x \neq n\pi$
$\arccos(x)$	1	0.54030230586813977	$x \notin (-\infty, -1) \cup (1, \infty)$
$\arcsin(x)$	0	0.8414709848078965	$x \notin (-\infty, -1) \cup (1, \infty)$

Table 6.2: **Part of a Look-up Table for Zeros**

So, for the example 6.1.1 above, knowing that the function $\log(x)$ has a singularity at $x = 0$, and that $x = 0$ is in the co-domain of $h(x)$, means we can search for the values of x responsible for the singularities. So we perform $g^{-1}(h^{-1}(0))$ to get the desired set of singular points.

This algorithm is implemented as `singularitiesOf` in package `dO1AgentsPackage` (see Appendix A.1).

6.1.2 Finding Weight Functions

There are three types of weight function of interest:

- Algebraic – of the form $\frac{1}{(x-a)^c}$ giving rise to a singularity at a of degree c. We

are thus looking for integrals of the form

$$\int_{\alpha}^{\beta} \frac{g(x)}{(x-a)^c} \quad dx \quad | \quad a \in [\alpha, \beta]$$

- Logarithmic – of the form $\log(x-a)$ giving rise to an essential singularity at a i.e. an integral of the form

$$\int_{\alpha}^{\beta} \log(x-\alpha)\log(\beta-x)g(x) \quad dx$$

- Trigonometric – of the form $\cos\omega x$ or $\sin\omega x$.

The computational agent for each essentially uses a pattern-matching algorithm looking for the operator within an expression, although there are slight differences.

An Axiom `Expression` is considered as a list of kernels, each of which have an operator and argument. If either `log` or `%power` appear as the operator of one of these kernels and a singularity exists, further investigation is undertaken to establish if the argument contains either one of the end points of the range of integration or is of the form $\frac{1}{(x-a)}$ where a is within the range of the integral.

If `cos` or `sin` appears as the operator of one or more of the kernels, it is necessary to find if it signifies a weight function. This is made more difficult because the internal representation of the expression may not be of the form **weight** $\times$ **subexpression**. In such a case, it looks for a common factor of the required type[8]. If there are two or more such weights it chooses the one with highest frequency ω.

A further requirement is to extract the weight from the integrand returning the weight, the transformed integrand and an indication of the type of weight found. These computational agents are implemented as

`exprHasAlgebraicWeight`

[8]Whilst it is eminently reasonable when performing formal integration on, e.g.

$$\int_{\alpha}^{\beta} \cos(\omega x)f(x) + g(x)\ dx$$

to split the integral into an oscillatory part and a non-oscillatory part,

$$\int_{\alpha}^{\beta} \cos(\omega x)f(x)\ dx + \int_{\alpha}^{\beta} g(x)\ dx$$

there would be no benefit to doing this for numerical integration. The extra cost of splitting and numerically integrating the two functions over the same range would by far outweigh any advantage from using a special routine on the oscillatory part.

`exprHasLogarithmicWeights`
`exprHasWeightCosWXorSinWX`

in package `d01WeightsPackage` (see Appendix A.1).

6.1.3 Miscellaneous Agents

The computational agent `functionIsOscillatory` estimates the number of zeros in the integrand. There are two cases. If there is a `sin` or `cos` term, the algorithm considers the range of its argument as a multiple of 2π. Otherwise it uses a quick, but dirty, method whereby it evaluates the integrand at 30 random points within the range and considers the number of sign changes.

Other agents include `rangeIsFinite` which tests the endpoints of the range for infinities and `problemPoints` which is a quick version of `singularitiesOf` for functions which are of the type, or can be coerced to the type, `Fraction Polynomial DoubleFloat`. It checks the denominator polynomial for zeros using Sturm sequences [Davenport *et al.*, 1988, pp 124–128] [Collins & Loos, 1983]. This also forms part of the algorithm for `singularitiesOf`.

6.2 Differential Equations

There are a number of factors which affect the choice of a suitable numerical solver for a particular initial value ODE problem. Foremost amongst these is the problem of stiffness, that is that the solution evolves on different time scales [Prothero, 1976]. Other attributes could be the stability of the solution, the cost of evaluating the ODE or its Jacobian, or the accuracy required of the solution.

Unfortunately, none of these have distinct answers – the ODE could be *partially* stiff or *slightly* unstable. Also, these can conflict with each other i.e they interact competitively. "For example, if a system is stiff, but very large, the problem may degrade the performance of a stiff code to the extent that a non-stiff code is preferable" ([Lucks & Gladwell, 1992, p. 12]). This conflict will be handled by the inference mechanisms and the knowledge rules in sections 7.1 & 8.2. This section deals with the computational agents to provide information (in the form of a normalised value) on the *intensity* of an attribute [Dupée & Davenport, 1996]. They are thus called *Intensity Functions* and have the postfix `IF`.

6.2.1 Testing for Stiffness and Stability

In dynamics, chemical engineering and electronics, the study of physical systems and the modelling process produces differential equations which may have solutions with both rapidly and slowly decaying components. One such analogy is to a complex vibrating mass-spring system where the springs are of wildly different stiffness coefficients. In trying to obtain a numerical solution to a system of such equations, this differing behaviour causes major problems for some standard algorithms such as Runge-Kutta and Adams methods since the direction of the solution vector at any particular time is swamped by the local behaviour. Certain alternative methods have been found to deal with this problem. It is therefore necessary to identify when such a system has a degree of stiffness.

Each system of differential equations has an associated Jacobian matrix which can be evaluated at, or near, the initial values. The eigenvalues of such a matrix give an indication of the stiffness of such a frozen system[9] [Lambert, 1973, pp 228–236]. It is assumed in many models of dynamic systems that the stiffness ratios are constant throughout the range. This may not be the case in more arbitrary systems, where one might be advised to use alternative, but more expensive, methods. [Dekker & Verwer, 1984, pp 10–12]. However, this is beyond the scope of current work.

So the computational agent `stiffnessAndStabilityOfODEIF` calculates symbolically the Jacobian matrix (if the system is not too large i.e. $< 12 \times 12$) and its real eigenvalues. Since symbolic methods can be expensive in finding complex eigenvalues, the computational agent uses a technique that would seem perverse should it be considered by a numerical analyst. Should the number of distinct real eigenvalues not equal the dimension of the Jacobian (or one less than the dimension of the Jacobian since complex eigenvalues can only occur in pairs)[10], the algorithm calls on the appropriate NAG Fortran Library routine (`F02AFF`) to calculate them numerically. This can be considered since, once the link to the NAG Library is in place, its use will not be as expensive as it would be either to create the code manually or to continue to use symbolic methods. This, in one way, can be thought of as a form of recursivity — creating a Fortran program to find out which form of Fortran program should be created to solve the problem.

[9] If two or more of the eigenvalues have negative real parts, an estimate of the stiffness is the ratio of the most negative to the least negative

[10] If some of the eigenvalues have multiplicity > 1 such that the number of eigenvalues is less than that required, the algorithm cannot distinguish them from complex eigenvalues without calculating the eigenvectors. This could be too expensive if the calculation is attempted symbolically.

Algorithm 6.1 (Computational Agent)

function *stiffnessAndStabilityFactor*
  if *system is not too big* **then**
    compute *real eigenvalues*
  if *number of eigenvalues is sufficient*[10] **then**
    compute *stiffness ratio*
    output *stiffness ratio (system is stable)*
  else
    call *f02aff*
    if *imaginary coefficients all zero* **then**
      output *stiffness ratio = zero (system is stable)*
    else
      compute *eigenvectors*
      compute *stiffness ratio*
      compute *stability factor*
      output *all*

At the same time as investigating the stiffness of the ODE, it calculates a stability factor which is the proximity of the negative eigenvalue closest to the imaginary axis. This may or may not be one of the eigenvalues responsible for the stiffness coefficient previously calculated. The effect of this proximity is a system with a rapid sine or cosine factor in its solution. This can also have a detrimental effect on certain routines, in particular those implementing the BDF method and, to a smaller extent, the Adams method.

6.2.2 Other Agents

The other computational agents associated with ODEs are primarily concerned with the *cost* of calculation and evaluation of the ODE. The function `systemSizeIF` returns a value in the range $[0, 1]$ as a function of the number of first-order equations in the system (e.g. a system of 20 equations is considered neutral and would give a value of about 0.5, many more than that would give a higher value); `expenseOfEvaluationIF` considers the cost of evaluation of the ODE as a function of the number of mathematical operations required (a neutral value would be given by the equivalent of about 200 multiplications); `accuracyIF` returns a value for the accuracy requirement and `intermediateValuesIF` returns a value for the number of intermediate values that the

user requires solutions, usually for later plotting or further analysis. Each of these affect in some way the possible optimum step-sizes for the numerical solver. These are all implemented in the package `d02AgentsPackage`.

For Partial Differential Equations, one of the essential requirements for some solvers is whether the system is elliptic. The computational agent `elliptic?` in package `d03AgentsPackage` uses the facilities provided by this expert system to utilise a number of numerical optimization techniques for testing for this attribute.

6.3 Optimization

An important question in mathematical modelling concerns the problem of finding the location of the local minima or maxima of a function [Cheney & Kincaid, 1985]. Numerical optimization techniques, specifically *minimization*, are often required for large problems, usually involving a list of constraints. It is not always apparent what sort of problem the user has been presented with, so most of the computational agents are designed to aid categorisation of the problem and the constraints.

6.3.1 Categorising the Optimization Problem

The computational agents `simple?`, `linear?`, `quadratic?`, and `nonLinear?` contained in the package `e04AgentsPackage` test both functions and constraints for these attributes. There is also a computational agent for testing whether a particular (unconstrained, univariate)[11] problem could be put in the form of a sum of squares, by considering its square-free factorisation[12], and thus be applicable to more efficient numerical algorithms.

[11] The computational agent is required to test whether a problem that the user *assumes* is of a type usually solved using standard minimisation techniques, but is probably better solved using least-squared approximation. Due to the difficulty of providing in Axiom an efficient mechanism for testing multivariate expressions, it will therefore be a single univariate function.

[12] Axiom can only consider the square-free factorisation of expressions that can be put into polynomial form, so it is possible that not all sums of squares can be identified.

6.3.2 Sorting Constraints

Since some routines require constraints to be entered in order of degree, the computational agents above can be used in a sorting algorithm[13] to order the constraints. This relieves the user from the task of always remembering that the numerical routines require the correct order. Since the constraints have the internal representation of three lists (the constraint functions, the upper bounds and the lower bounds), care is taken to ensure that all three lists are ordered simultaneously. This is really a usability issue, but an important one, considering that the aim is to create a more intelligent interface to these routines.

6.4 Conclusion

I have described a number of different computational agents, some of which use heuristics while others use deterministic methods. In general, these have been designed to perform their task with reasonable efficiency. Of course, sometimes this means that the answers they give may contain errors.

For example, there does not exist a perfect algorithm for calculating all the singularities of a function and even if one existed, it would be likely to take a prodigiously long time. The difficulty arises with expressions of the form $\frac{1}{f(x)+g(x)}$ where either $f(x)$ or $g(x)$ is an exp-log function and the other is non-constant. In this case the algorithm I have implemented may not find the singularity. In such a case where the singularity exists, there are three possible outcomes: the chosen routine may fail so an alternative is used which succeeds; the chosen routine succeeds since the singularity is either removable or not significant; or all routines will fail.

There are cases where the test used for stiffness of a set of ODEs will give misleading results, especially if the stiffness is apparent only on a portion of the range. Fortunately, since the routines used are adaptive, the effect is unlikely to be disastrous. Another possible cause of some inefficiency is if there are real eigenvalues of the Jacobian with such multiplicity that it forces the computational agent to perform further analysis. This extra analysis is not onerous and will not slow the process greatly.

The test for oscillations of an integrand could fail if all the oscillations are contained in

[13] I have implemented a bubble-sort since simplicity of algorithm design is important and, should there be a large number of constraints such that bubblesort is not optimal, the time spent ordering the constraints is relatively small compared to the time performing the numerical stages.

a small part of the range. Should that be the case, a routine specifically for oscillatory functions would not be entirely appropriate.

So, where there are possible traps, I have endeavoured to minimize their effects so as to provide as consistent a set of computational agents as is reasonable.

Chapter 7

The Knowledge Base

Knowledge is the conformity of the object and the intellect.

Averroës (1126 - 1198) Destructio Destructionum

The major part of the knowledge base is concerned with the possible methods that can be used to solve particular problems posed by the user. These fall neatly into the four 'chapters' of numerical integration, ordinary differential equations, partial differential equations and optimization. For this reason, the method domains are divided amongst the four Axiom Categories (see Appendix B.1). Each of these Categories provide a consistent structure, operations and interface to these methods implemented as Axiom Domains corresponding to §5.1, p. 34, 1 and 2.

Other parts of the knowledge base are concerned with the current state of the knowledge gained about each problem (Dynamic Knowledge — §5.1, p. 34, 5(a) and 5(b)), and knowledge about basic elementary functions used by certain computational agents rather than the inference engine (§5.1, p. 34, 5(f)).

For ease of use, a further table is used which contains a list of available routines and their application areas, together with indications whereby, if a routine has already been found which has a good likelihood of being able to be used efficiently, a lazy evaluation mechanism is triggered (§5.1, p. 34, 5(c)). This Domain `RoutinesTable` also contains details of the `IFAIL` values and indications of possible fall-back strategies as well as initial values for measures which are altered or optimized in the Measure Domain.

These `IFAIL` instructions often only require the deletion of that particular routine from

the database and re-input the problem. Sometimes, however, it may be necessary to alter the required tolerance (either up or down).

The routine `d01ajf` is listed in this database as:

```
chapter= "Integration",
type= "One-dimensional finite",
domainName= "d01ajfAnnaType",
defaultMin= 0.4,
measure= 0.4,
failList =
  [[ifail= 1,instruction= "delete"],
   [ifail= 2,instruction= "delete"],
   [ifail= 3,instruction= "delete"],
   [ifail= 4,instruction= "delete"],
   [ifail= 5,instruction= "delete"],
   [ifail= 6,instruction= "delete"]]
```

The Domain also has a number of functions for both searching this database and modifying any entries.

7.1 Knowledge of Methods

An Axiom Domain has been created for each method or strategy for solving the problem. These method Domains each implement two functions with a uniform (method independent) interface:

measure: A function which calculates an estimate of suitability of this particular method to the problem if there is a possibility that the method under consideration is more appropriate than one already investigated.

If it may be possible to improve on the current favourite method, the function will call computational agents to analyse the problem for specific features and calculate the measure from the results these agents return. It also calculates any method-specific parameters, such as weight functions, points and types of possible discontinuities etc., for later use.

implementation: A function which may be one of two distinct kinds. The first kind uses the interface to the NAG Library to call a particular routine with the required

parameters. Some of the parameters may need to be calculated from the data provided before the external function call such as workspace parameters. It also makes sure that all the data are in the correct form i.e. that the parameters are of the correct types, external functions are properly named and specified and all parameters are in the correct order.

The other kind applies a "high level" strategy to try to solve the problem e.g. a transformation of an expression from one that is difficult to solve to one which is easier, or a splitting of the problem into several more easily solvable parts. This may thus enforce some recursion on the *measure* function.

For example, the Integration Domain `d01apfAnnaType`, a routine for calculating integrals where the integrand is of the form of Equation 6.1 (p. 42) contains the two functions `measure` and `numericalIntegration`.

Algorithm 7.1 (Method Domain d01apfAnnaType)

function *measure*
  initialise *c, d, l*
  compute *algebraic weights*
  if *integral has algebraic weights* **then**
    set *c, d*
  compute *logarithmic weights*
  if *integral has logarithmic weights* **then**
    set *l*
  if *no weights found* **then output**
    `[ 0, "d01apf: A suitable singularity has not been found"]`
  else compute *measure*
  output
    `[ measure, "Recommended is d01apf with c = " c ",`
    `d = " d " and l = " l ]`

function *numericalIntegration*
  $Fac1 \leftarrow$ **compute** $x - a$
  $Fac2 \leftarrow$ **compute** $b - x$
  # compute factors
  $fac \leftarrow Fac1^{c} * Fac2^{d}$
    if $l > 1$ **then**
      if $l = 2$ **then**
        $fac \leftarrow fac * \log(Fac1)$

```
    else if l = 3 then
      fac ← fac * log(Fac2)
    else
      fac ← fac * log(Fac1) * log(Fac2)
  # reduce integrand
  Fn ← Fn/fac
  f ← convert Fn to Fortran
  call d01apf
```

Within the function `measure`, it searches for algebraic and logarithmic singularities at the end points of the range of the integrand. If it finds any, this is signaled to the inference mechanism as a positive 'measure' along with details of the singularities found.

If the expert system inferred that this was the most appropriate routine to perform the integration, the function `numericalIntegration` would translate the information thus gained into a form suitable for the NAG library routine. It has to separate the logarithmic and algebraic weights from the function before it is translated into a Fortran function. This is then passed directly to the Fortran routine together with the other parameters.

Another Integration Domain of considerable interest is `d01TransformFunction`. This is designed to investigate the appropriateness and perform an algorithm to transform an infinite integral into either one or two finite integrals and thus allow the system to use a better range of numerical routines. Its `measure` function is:

Algorithm 7.2 (Method Domain d01TransformType)

```
function measure
  compute range
  if both ends infinite then
    call split
  if upper end infinite then
    if lower end is positive then
      call transform
    else
      call split
  if lower end infinite then
    if upper end is negative then
```

```
    call transform
  else
    call split
```

This calls on two local agents `split`, if the function needs to be split into two, and `transform` to transform it onto a finite region using the mapping $x \mapsto 1/t$.

```
local function transformFunction
  Mapping ← compute x = 1/t
  Integrand ← apply Mapping to Integrand
  simplify Integrand

local function transform
  compute range
  call transformFunction
  m ← call measure # Top Level Call
  output
    [ m, "The recommendation is to transform the function
                          and use " name, List of Hints ]

local function split
  compute range
  m1 ← call measure # Top Level Call
  call transformFunction
  compute range
  m2 ← call measure # Top Level Call
  m ← combine m1 and m2
  output
    [ m, "The recommendation is to transform the function
          and use " name1 " and " name2 , List of Hints ]
```

It places all this information in a list for later implementation (should it be required) within the function `numericalIntegration` which recursively calls a top level integrate function on the list:

```
function numericalIntegration
  for Hint in List of Hints repeat
    compute Integral
```

```
    call integrate # Top Level Call
    compute Result
    compute Error Estimate
  output All
```

7.2 Dynamic Knowledge Representation

Each method domain can call any of the computational agents it needs to calculate how appropriate that method is to the current problem. Some agents will be common to more than one method domain or problem category, whereas others may be specific to a single domain. For example, the computational agent `expHasAlgebraicWeight` is only used by the Domain investigating the usefulness of the integration routine `d01apf`, whereas `stiffnessAndStabilityOfODEIF` is used within all ODE method Domains.

For this reason, to minimize computation, each computational agent places a copy of the knowledge it has gained into a keyed table. The key for each item is the current problem specification. So before a particular computation agent performs its task, it checks to see if it has already been called with this same problem and therefore does not need to do any recalculation. It can also tell if this particular problem has already been investigated within the current session. This knowledge is used by parts of the inference machine and as a significant part of the explanation process, the current values of the table being presented as part of the output.

All other storage of dynamic data (§5.1, p. 34, 5(e)), such as results of previous problems, are handled by the Axiom indexed buffer. For example, the n^{th} result is accessed as `%%`(n) [Jenks & Sutor, 1992] c.f. Appendix A.

Chapter 8

Measure Functions

Est modus in rebus.

Horace (65 - 8 B.C.) Satires

In this expert system we are faced with evaluating potential strategies for obtaining solutions to a set of mathematical problems. This entails measurement of various attributes of each problem for use as *evidence*. As described in [Dupée, 1996], we require a sound theoretical basis for this measurement and therefore use *belief functions* or *measure functions* to give values to the effectiveness of each strategy.

The requirement is to assign to each method some measure which tells us how effective the method is likely to be given the current problem. The aim is thus to get numerical measures for the effectiveness of each of a number of possible methods for automatic comparison. If the method applications were disjoint, for example, if for each problem only one method were possible, such measurement would be trivial. Also, if all methods were simple i.e. not likely to include multiple strategies, and attributes of the given problem did not in any way conflict, Bayesian methods would be appropriate. However, as will become apparent, these complications appear and require alternative approaches.

Sometimes we use strategies which are themselves multiple strategies i.e. we wish to split the problem into a number of pieces and wish to give appropriate values to them for comparison with other multiple and singleton strategies. This process forces a structure on the underlying singleton values.

Other strategies may have attributes which conflict in some way. We therefore have to

use functions describing the compatibility of each attribute *in combination* with each possible strategy.

Bayesian theory is restricted to singleton hypotheses and so cannot on its own be applied where multiple strategies are possible. For this reason, a more respectable solution is applying Dempster-Shafer theory ([Gordon & Shortliffe, 1983; Paris, 1994]) whereby we can assign measures to any number of subsets of the set of strategies. However, such a set of subsets does not normally include multiple copies of the same strategy.

[Gordon & Shortliffe, 1985], [Pearl, 1986] and [Shafer & Logan, 1987] show us how we can construct belief spaces for evidential reasoning using Dempster-Shafer theory on a hierarchy of hypotheses. This is used as a basis for considering hierarchies of the complete belief space and some of the effects this necessitates on the system. The following section is a summary of the main theoretical ideas in [Dupée, 1996] which allows the use of limited recursion within Dempster-Shafer theory as a basis for the comparison of single and multiple strategies.

8.1 Dempster Shafer Theory with Multiple Strategies

Definition 8.1 *Let* Θ *be a* **Measurable Discrete Topological Space** *of the set of subsets of a set of methods* S*. This is the* **Dempster-Shafer frame of discernment.** *So the set of hypotheses is* Θ*. It is not discounted that two or more members of* S *represent the same method.*

Example 8.1.1 *Let the set of methods* $S = \{A, B, C\}$[1]*. Also, let* A *and* B *represent the* same *method and* C *represent* two copies *of method* A*. We can construct* Θ *as all subsets of* S*. This set* Θ *contains the subsets* $\{A, B\}$ *and* $\{C\}$*; each, ostensibly, representing the same strategy i.e. using method A twice. We will have to reconcile these and judge their respective values within our proposed model.*

---

[1] We might wish to consider that A and B are both the general integration method `d01ajf` and C represents the method `d01amf`, which splits the function and uses `d01ajf` on each section. S therefore represents all combinations of these i.e. includes the combination of performing the splitting *outside* the routine (as implemented in the Method Domain `d01Transform`) and using `d01ajf` twice, as well as the *single* method `d01amf` which splits the function *inside* the routine

What we propose to do is to work theoretically on multiple levels i.e. treat singleton strategies differently from multiple strategies. To do this, we consider S as the set of methods *without copies.* An element of Θ which represents a multiple method strategy, instead of using the singleton elements within Θ, *creates a number of copies of* Θ. In the terminology of Shafer ([Shafer, 1976]), this is a *refinement* of Θ such that an element of Θ is recursively refined as being one or more copies of Θ.

The extension of the Dempster-Shafer frame of reference to include multiple topologies has an effect on the inference architecture, since we are now having to work on a number of different levels. This forces us to maintain levels of belief in Θ even though we can completely assign values to all singleton elements of Θ. The effect can be seen in an example.

Example 8.1.2 *Let S be the set of strategies $\{M_1, M_2, \ldots M_n\}$ where M_1 is a strategy representing a number of any of the methods $M_2 \ldots M_n$. The* basic probability assignments (b.p.a.) *for the singleton methods of Θ are calculated in the usual way, but the* b.p.a. *of M_1 is calculated from one or more* copies *of Θ, say $\Theta_1, \ldots \Theta_m$, by combining the largest* b.p.a.s *of the $\Theta_1, \ldots \Theta_m$.*

This therefore splits the original problem into two or more pieces and a b.p.a. *obtained for each of these separate subproblems.*

The effect of this is to cause difficulties with consistency of the system. This can be solved using a further normalisation. Now that we have the foundations of a hierarchical system, we can investigate some of the features and peculiarities the system provides.

Example 8.1.3 *Let S be as above and M_2 represent a singleton method which splits the problem and uses a particular method on each sub-problem.*

Furthermore, let the remaining $M_3 \ldots M_n$ be subdivided into two groups i.e. specific *and* general. *A* specific *method is one that is specific to a single type of problem. A* general *method is one which can be addressed to many different types of problem (with, perhaps, a differing degree of reliability).*

Let us consider the b.p.a. *for each of M_1 and M_2, i.e. we have a problem which can be solved using these two strategies. If the individual methods used both by M_1 and M_2 are* general *methods, the* cost *of using M_1 is likely to be higher so the system should give a higher* b.p.a. *to M_2. However, if* one *or* both *of the methods selected for use*

by M_1 *is* specific, *and* specific *methods are always preferred, a higher* b.p.a. *should be allocated to* M_1.

Using **Dempster's Rule of Combination**, *this would only be possible if the* b.p.a. *of* specific *methods (having the highest* b.p.a. *in the sub-topologies above) is greater than* 0.5 *and all* general *methods have a best* b.p.a. *of less than* 0.5.

Since there can be no difference between the pattern of the topology Θ *and the copies of* Θ *used to calculate the* b.p.a. *of* M_1, *this structure must be applied throughout. Furthermore, the maximum* b.p.a. *for* M_2 *must be finely judged as being slightly greater than the maximum* b.p.a. *for the combination of the singleton* general *methods it represents.*

Whilst I have used only a single multiple strategy extending the topologies, it is possible, using the same reasoning, to expand the example to have more than one strategy with this feature.

8.2 Conflicting Evidence and Lucks/Gladwell Measures

Where we have conflicting evidence, Dempster-Shafer theory can be applied to calculate the plausibility of each singleton method. In combination with possible multiple methods, the normalisation process is more complex since we must always maintain belief in Θ to allow for the extended topologies.

However, the implementation we have used deserves a little explanation. In keeping with the system recommended by Lucks & Gladwell ([Lucks & Gladwell, 1992]) we have introduced four types of functions:

measurement functions[2] quantifying the degree of presence of features in an input elementary problem;

intensity functions conversion of the measurement of features onto a standard scale;

compatibility functions describing relationships between the degree of presence of features and the behaviour of the inner workings of the methods;

aggregation functions describing the overall behaviour of a method based on the aggregate effect of the individual problem features.

[2]These are not the same as the measure functions of this chapter but are the initial measurements of the individual attributes.

The compatibility functions are used as the input to our D-S model which is implemented within the aggregate functions. We can therefore economise on code by using dynamic table lookup for values obtained for the intensity functions. The behaviour of individual methods under the influence of various features is an area that takes as its basis the *judgement* of Numerical Analysis "experts" whether that be from documentation or alternative sources. However, its assessment of the suitability or otherwise of a particular method to a particular problem is reflected in a single normalised value facilitating the direct comparison of the suitability of a number of possible methods or strategies.

8.3 Application

Given the example of assessing the applicability of numerical integration routines, we can think of these methods as falling into three subgroups – those that implement a general strategy which can be applied to a large subset of integrals; those that implement a specific strategy for applying to particular subsets of integrals i.e. of those functions which reveal a particular attribute or set of attributes; those that implement a number of strategies e.g. those that split the function into two and perform different strategies on each part. However, it is of extreme benefit to maintain a consistent interface to each of these methods and each must be considered equally.

So we have, for example, a routine `d01ajf` which implements a strategy which can be addressed to a wide variety of different classes of integrals, but which may fail to work, or give an inaccurate result, under certain difficult conditions. We have a routine `d01apf` which is ideal for use in situations where the values of the integral at the end points of integration are undefined or uncalculable, but which are of little or no benefit in other cases. We also have routines such as `d01amf` and `d01transform` which, should one or both of the end points of integration be infinite, will split the function and transform each part onto a finite region before implementing one or more of the other routines. The routine `d01amf` performs this internally (hard-wired into the Fortran code) using a general routine for the implementation and `d01transform` does the splitting and transformation externally (in the Axiom interface to the NAG routines) and can then implement any of the other routines depending on the attributes it finds.

So specific routines such as `d01apf` give a positive *b.p.a.* if the specific attributes or combination of attributes is present; `d01ajf` and other general routines give a positive

b.p.a. *unless* certain difficult conditions prevail; `d01amf` gives a positive *b.p.a.* if the integral is finite or semi-infinite (since the details of transformation and implementation is hidden, no other analysis can take place) whilst `d01transform` performs full analysis of the different parts of the integral and calculates its *b.p.a.* from the *b.p.a.*s of the individual routines which should be considered appropriate for each separate part. Therefore, if specific routines are appropriate, this strategy should be applied in preference to `d01amf`.

Where the expert system is attempting to assess possible routines for solving a system of ODEs, the Lucks/Gladwell compatibility functions are used as the input to the D-S frame. The complete measurement process is thus kept within the Method Domains.

Chapter 9

Inference Mechanisms

The question now is, to know whether the mind has made this inference right or no; if it has made it by finding out the intermediate ideas, and taking a view of the connection of them, placed in due order, it has proceeded rationally, and made a right inference.

J. Locke (1632 - 1704) An Essay Concerning Human Understanding

9.1 Inference Packages

The choice of method domains and computational agents means that there is a requirement only of a small and relatively simple inference engine (c.f. §5.1, p. 34). The job of this inference engine is to control the process of choosing a possible method and performing any recovery mechanism should that be required. It may also be required to transform both input and output into some standard form, either for use by the system or for further possible use by some other agent.

Four Axiom packages have therefore been created, one for each problem area, with similar functionality which additionally act as the command line interface to the package and provide the remaining sections of the [Newell & Simon, 1972] Information Processing System §5.1, p. 34, 3 and 4. For example, the package for integration contains functions as below.

Algorithm 9.1 (Inference Machine for Integration)

```
function measure
   select database entries for relevant routines
   initialise best measure so far
   for each relevant routine repeat
     if possible measure > best measure so far then
       call measure function of method
       if measure > best measure so far then
         assign measure to best measure so far
         assign hints to database
   output result of best measure found

function integrate
   convert data to correct form
   call measure
   call integrate function of best measure
   if integrate function fails then
     call recovery procedure
   convert output to correct form
   output result

local function recover
   while result is not satisfactory do
     store result
     if recommendation is to repeat integration with
          changed parameters then
       change parameters
       call integration routine again
     else if only one routine in database
          result is best we can do
     else
       delete routine from database
       call integrate # Top level function
```

By reducing the size and complexity of the necessary inference mechanisms, it is easier to see the relations between the expert system and the definition Newell and Simon use

of the decisions to be made within a problem-solving information processing system :

1. At a knowledge state (a node in the problem space), to select an operator to be applied.
2. At a new knowledge state, to determine whether problem solving shall continue from this state or not.
3. At a knowledge state, to determine whether the knowledge state shall be remembered, so that return can be made to it at some later time.
4. At the decision to abandon a knowledge state, instead of continuing to search from it, to select another knowledge state in the backup state.

[Newell & Simon, 1972, p. 826]

9.2 Recovery Procedures

Indication that a problem has been encountered during the numerical stages of the computation is provided by the Naglink in the form of a non-zero value of the parameter `ifail` which corresponds to the Fortran parameter of the same name returned by the NAG library routine. In the NAG Fortran library documentation pages referring to each routine there is a list of failure codes, each with their possible causes and recommendations. For example, the error failure codes for the ordinary differential equations routine using the Adams-Bashforth-Moulton method `d02cjf` are listed by the on-line help as:

```
*     D02CJF
D. Error Indicators and Warnings

  Errors detected by the routine:

  If on entry IFAIL = 0 or -1, explanatory error messages are
  output on the current error message unit (as defined by
  //X04AAF//).

  IFAIL = 1
    On entry, TOL <= 0.0,
    or        N <= 0,
    or        RELABS <> 'M', 'A', 'R' or 'D',
    or        X = XEND.
```

```
IFAIL = 2
  With the given value of TOL, no further progress can be made
  across the integration range from the current point x = X. (See
  Section 8 of the routine document in the NAG Fortran Library
  Manual for a discussion of this error exit.) The components
  Y(1),Y(2),...,Y(N) contain the computed values of the solution
  at the current point x = X. If the user has supplied g, then no
  point at which g(x,y) changes sign has been located up to the
  point x = X.

IFAIL = 3
  TOL is too small for //D02CJF// to take an initial step. X and
  Y(1),Y(2),...,Y(N) retain their initial values.

IFAIL = 4
  XSOL has not been reset or XSOL lies behind X in the direction
  of integration, after the initial call to OUTPUT, if the OUTPUT
  option was selected.

IFAIL = 5
  A value of XSOL returned by OUTPUT has not been reset or lies
  behind the last value of XSOL in the direction of integration,
  if the OUTPUT option was selected.

IFAIL = 6
  At no point in the range X to XEND did the function g(x,y)
  change sign, if g was supplied. It is assumed that g(x,y) = 0
  has no solution.

IFAIL = 7
  A serious error has occurred in an internal call. Check all
  subroutine calls and array sizes. Seek expert help.
```

It is immediately noticeable that if, for example, `IFAIL` was returned with either value 1, 4, 5[1] or 7 that, on the proviso that the Method Domain was correctly implemented, this would signify some internal error of either Axiom, the Axiom-NAG link (nagd or nagman) or of the NAG library routine itself. Since it would not be possible to discern which of these is the cause, the most reasonable move would be to disregard the output from the routine and re-submit the problem.

But not all `IFAIL` values fall into this category. Therefore there are a number of different recovery procedures which require implementing:

[1] `XSOL` is a parameter of the automatically created Fortran subroutine for the storage of intermediate values.

- If there is a catastrophic failure i.e. if the routine fails to provide any result, the routine is to be removed from the database and a different routine should be chosen.
- If an alteration of, say, the error requirements is suggested and there is no other indication of possible failure, i.e. by altering such parameters the result can be obtained, such action should be taken and the routine re-implemented.
- If the best possible error value returned by the routine is higher than that requested, a further routine is to be chosen and, if successful, the results output alongside indications of the failed routine and its results.
- If the indication is only a warning or some other information which does not indicate any misgivings in the given result, no further action is required.

This information has been incorporated within the table of routines section of the knowledge base (see §7) as:

```
chapter= "ODE",
type= "IVP",
domainName= "d02cjfAnnaType",
defaultMin= 0.7,
measure= 0.5,
failList =
  [[ifail= 1,instruction= "delete"],
   [ifail= 2,instruction= "decrease tolerance"],
   [ifail= 3,instruction= "increase tolerance"],
   [ifail= 4,instruction= "delete"],
   [ifail= 5,instruction= "delete"],
   [ifail= 6,instruction= "no action"],
   [ifail= 7,instruction= "delete"]]
```

Since some IFAIL values of a number of routines could represent a range of possible failures, there may be times when the rejection of a routine could be considered a little drastic except that further analysis of why the routine has failed is likely to be complicated and not a process which could be implemented automatically.

Certain other failures would be due to incorrect array sizes or other mistakes from improper parameters or a poorly specified problem. This should not normally occur

if the Method Domains are correctly implemented. If this is not the case, the correct strategy is to remove that routine from the database and restart the process.

Chapter 10

Explanation

> *Modern-day computers are amazing pieces of equipment, but most amazing of all are the uncertain grounds on account of which we attach any validity to their output.*
>
> *It is not only the programmer's task to produce a correct program but also to demonstrate its correctness in a convincing manner.*
>
> *E. Dijkstra, et al.* Structured Programming

The "fundamental goal of an explanation is to enable a program to display a comprehensible account of the motivation for all its actions" [Davis & Lenat, 1982]. As intimated in §5.5 p. 38, the level of belief the user could have that an expert system is successfully mirroring the thought processes of an expert is entirely dependent on the type and quality of any explanation it gives and on tailoring any explanation to the knowledgeability or requirements of the user. Obviously, if the user wishes to use the expert system as a "black box", no explanation is necessary or required. In such a case, it would be considered superfluous and thus a mistake should one be provided. It would, of course, also be a mistake if too little information is provided such that the reasoning behind the choice of routine is left unclear at any time.

It would also be unnecessary to provide what John Locke would term the *essential properties*, only *accidental properties*[1]. For example, if we were asked to explain a choice of routine for solving a set of Ordinary Differential Equations, we are not expected

[1] Locke, himself, based his ideas of these properties on Aristotle, who spoke in terms of Plato's 'Universals', and on Descartes' primary and secondary qualities. [Trusted, 1981]

to identify all characteristics of ODEs – only those which have some bearing on the final choice. It is also necessary, in general, to couch the explanation in terms of the application domain so as not to be too imprecise.

The explanation process has two identifiable and discrete tasks – to clarify and to justify. Clarification is a description of what routine is chosen and the properties pertaining to the problem which have been identified or quantified. Justification of the choice of routine requires identification of which properties are important as well as some ideas about why the particular routine is thought to be better than others.

10.1 A Hierarchy of Explanation

There are a number of aspects of this expert system which must form part of the explanation process.

- What routine was used? i.e. the name of the method or strategy.
- Why was this method chosen? i.e. what attributes were found which make this method better than others?
- Why were other methods not chosen?
- On what basis were these decisions made? This information is of a different nature to those above in that it is non-method-specific.

The nature of the Axiom type `Result`, which is the default return type from a NAG routine, has a natural hierarchy. This can thus be used to provide the framework for the explanation. Thus, the return type from ANNA, also a `Result`, contains two extra fields specifically for the explanation mechanism. These are labelled `method` and `attributes` as shown in the example previously used in §4.3.3 on page 30:

```
(1) -> ans := integrate(1/(1.0+3*sin(t)^2),0..2*%pi)

   (1)
   [iw: Matrix(Integer), abserr: DoubleFloat, w: Matrix(DoubleFloat),
    ifail: Integer,,result: DoubleFloat, method: Result,
    attributes: List(Any)]
                                                    Type: Result
```

The non-method-specific information, i.e. the results of computational agents, can be accessed directly as:

```
(2) -> ans.attributes

   (2)  [Continuous at the end points,The range is finite,[]]
                                                        Type: List Any
```

This shows that the integrand shows no evidence of singularities at the end points or internal to the range and that the range of integration is finite. These attributes are not specifically required by any one method in assessing its appropriateness to the problem and thus separated from method-specific information.

The field **method** contains a number of sub-headings:

```
(3) -> ans.method

   (3)
   [nameOfRoutine: String, other: Result, allMeasures: List(String),
    bestMeasure: Float]
                                                        Type: Result
```

The sub-field **allMeasures** gives much insight into the measurement process and thus the inference mechanism:

```
(4) -> qelt(ans.method,allMeasures)

   (4)
   ["Trying One-dimensional finite integration routines",
    "d01aqfmeasure: 0.0 - d01aqf: A suitable weight function has not
       been found",
    "d01anfmeasure: 0.0 - d01anf: A suitable weight has not been found",
    "d01ajfmeasure: 0.4 - The general routine d01ajf is our default",
    "d01akfmeasure: 0.0 - d01akf: The expression shows little or no
       oscillation",
    "d01apfmeasure: 0.0 - d01apf: A suitable singularity has not been
       found",
    "d01alfmeasure: 0.0 - d01alf: A list of suitable singularities has
       not been found"
    ]
                                                  Type: List String
```

The sub-field **other** provides specific information on, for example, weight functions if any have been found, transformations if they have been used and method-specific parameters etc.

Much of this structure can be seen automatically if the command

```
showScalarValues true
```

is used e.g.:

```
(6) -> a := integrate((exp(-x^3)+exp(-3*x^2))/sqrt(x), 0.0..%plusInfinity)

   (6)
   [
     abserr: 2.69960156338737e-08,

     result: 3.23287256251958,

     method:
       [nameOfRoutine: "d01TransformFunctionType",
        other:
           [d01transformextra:
             List
              Record
                 :(str,String),
                 :(fn,Expression(DoubleFloat)),
                 :(range,Segment(OrderedCompletion(DoubleFloat))),
                 :(ext,Result)],
        allMeasures: List(String),
        bestMeasure: 0.6086956521 7391304348],

     attributes: List(Any),

     d01ajfAnnaTypeAnswer:
       [iw: Matrix(Integer), abserr: 2.6995941792608e-08,
        w: Matrix(DoubleFloat), ifail: 0, result: 0.085813447681579,
        method:
          [nameOfRoutine: "d01ajfAnnaType",
           other: [],
           allMeasures: List(String),
           bestMeasure: 0.4],
        attributes: List(Any)],

     d01apfAnnaTypeAnswer:
       [iw: Matrix(Integer), abserr: 7.38412656787854e-14,
        w: Matrix(DoubleFloat), ifail: 0, result: 3.147059114838,
        method:
          [nameOfRoutine: "d01apfAnnaType",
           other: [d01apfextra: List(DoubleFloat)],
           allMeasures: List(String),
           bestMeasure: 0.7],
        attributes: List(Any)]
   ]
                                                          Type: Result
(7) -> qelt(a.method,allMeasures)

   (7)
   ["Trying One-dimensional infinite integration routines",

    "d01amfmeasure: 0.5 - d01amf is a reasonable choice if the integral
```

```
    is infinite or semi-infinite and d01transform cannot do better than
    using general routines"
     ,
    "d01asfmeasure: 0.0 - d01asf: A suitable weight has not been found",

    "d01transformmeasure: 0.609 - The recommendation is to transform
    the function and use d01apfAnnaType and d01ajfAnnaType"
]
                                                        Type: List String
```

Here, the method chosen is to use the method `d01transform` to split the integrand and transform the infinite part onto a finite domain and use two different routines to perform the integration. ANNA returns the result, error estimate and method information as well as the individual output from the two constituent calls. The method information contains details of the transformation used:

```
(8) -> qelt(qelt(a.method,other),d01transformextra)
```

```
   (8)
   [
                                        2          3
                                    - 3.0x     - 1.0x
                                   %e       + %e
     [str= "d01apfAnnaType", fn= ----------------------, range= 0.0..1.0,
                                          +-+
                                         \|x
      ext= [d01apfextra: List(DoubleFloat)]],

                                     1.0       3.0
                                   - ---     - ---
                                      3         2
                                     x         x   +-+
                                (%e      + %e     )\|x
     [str= "d01ajfAnnaType",fn= ------------------------,range= 0.0..1.0,
                                          2
                                         x
      ext= []]]
```

```
              Type: List Record(str: String,fn: Expression DoubleFloat,
               range: Segment OrderedCompletion DoubleFloat,ext: Result)
```

This hierarchy fulfills all the requirements stated in §5.5 in that it provides sufficient clarification and justification of the inference process at the user's request.

Chapter 11

The HyperDoc Interface

HyperDoc is described in [Jenks & Sutor, 1992] as "an on-line tutorial and an on-line reference manual". It is the system used on the UNIX versions of Axiom for an active, windows-based high-level interface. The user can:

- Get help on how to use the HyperDoc system;
- Read about an extensive list of topics;
- Fill in templates for solving problems;
- Scan an on-line version of the Axiom reference manual;
- Look at examples of how Axiom can be used;

and much more. Its capabilities have been extended to provide an interface for ANNA. This involves providing a top-level link and a number of pages of information, templates and examples within some ordered structure.

The top level pages and information pages are written in HyperDoc's own mark-up language, some aspects of which are similar to HTML, the standard Hypertext mark-up language. Other pages, in particular those that perform some manipulation of input characters, have been written in "boot" code, an interface to the underlying lisp. Much of the graphics and special characters are provided as bitmaps, either colour or black and white.

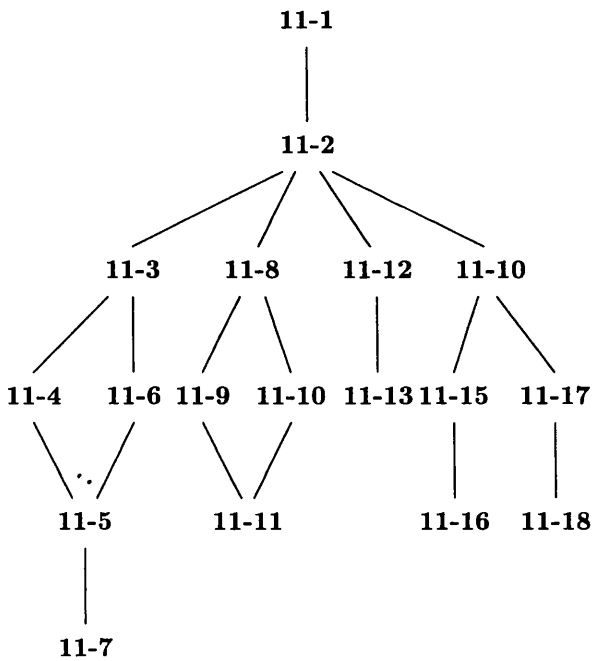
'Boot' is a generator of Hyperdoc pages — it can store, manipulate and export active data for the automatic formatting of HyperDoc pages and interfacing directly with Axiom, the lisp processor and the operating system. It is the programming language

which underpins all HyperDoc functions, whilst itself based on lisp and which itself can have embedded HyperDoc links. The 'boot interpreter' translates the programme code into lisp functions for further interpretation by Axiom sub-processes.

HyperDoc itself provides a number of formatting commands which allow the pages to be constructed easily with a consistent appearance. For each of these pages, parts, either graphics or text, can be defined as "active links" to other pages. This helps provide logical structure to the interface and opens the way towards providing help and tutorials. There is also the possibility of interfacing with either Axiom for executing commands, the underlying lisp for file access etc., or the operating system.

The use of 'boot' code provides a way of writing HyperDoc pages with integral lisp functions. These lisp functions can take data from the various parts of a document and manipulate them whilst building further pages. For example, if we have an n-dimensional system, we can enter n in an active area on one page and when the next page is opened, it can provide the requisite number of active areas for further data input, with any appropriate default values (c.f. Figure 11-9). A further active structure which can be used is the 'radio buttons', which can be either set *on* or *off* and can control whether or not certain input areas appear on a page, or what defaults appear in input areas.

The HyperDoc pages shown in Figures 11-1 to 11-18 relate to each other as:



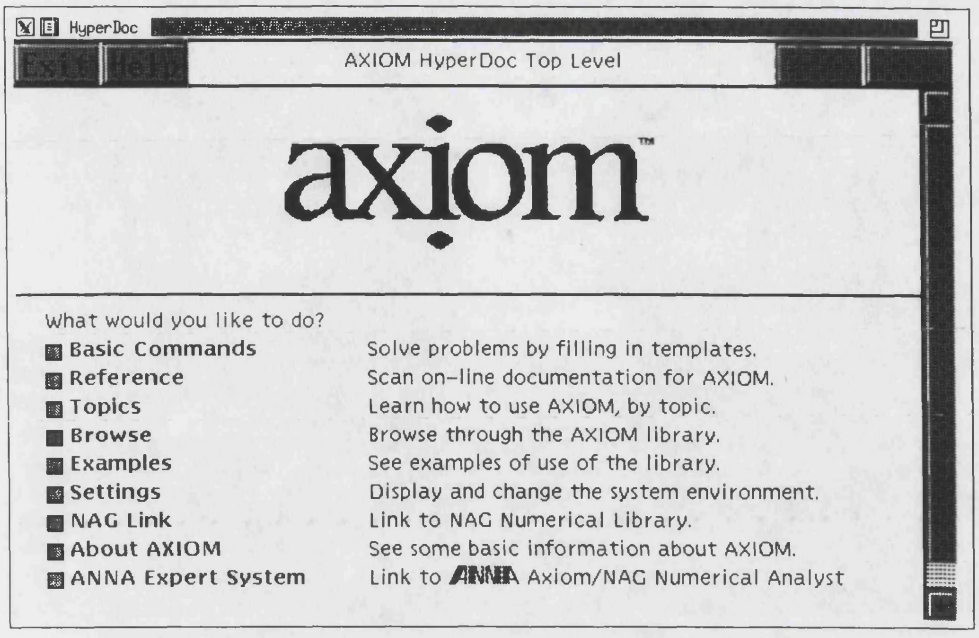


Figure 11-1: Axiom Top Level Page (modified for ANNA)

Further pages have been written to provide help in the use of ANNA and also provide explanations of the intricacies of why ANNA has be created and what it does. It also gives a few hints and examples on how ANNA can be used interactively from the command line and within composite Axiom programs.

There is also a section, with examples, on the use of the computational agents and how these can be called directly from an interpreter window.

In the process of creating these pages, a number of design decisions were made. Some aspects were, in general, dictated by the requirements that the pages should look similar to those already created for use within the Axiom system, as regards fonts, bullet points and headings. However, since ANNA provides a simplified view to numerical software, some freedom on page layout and structure was allowed. This is particularly apparent in the input pages for ODE and optimization problems where, for example, constraints and the bounds on those constraints are logically organised (see Figure 11-16).

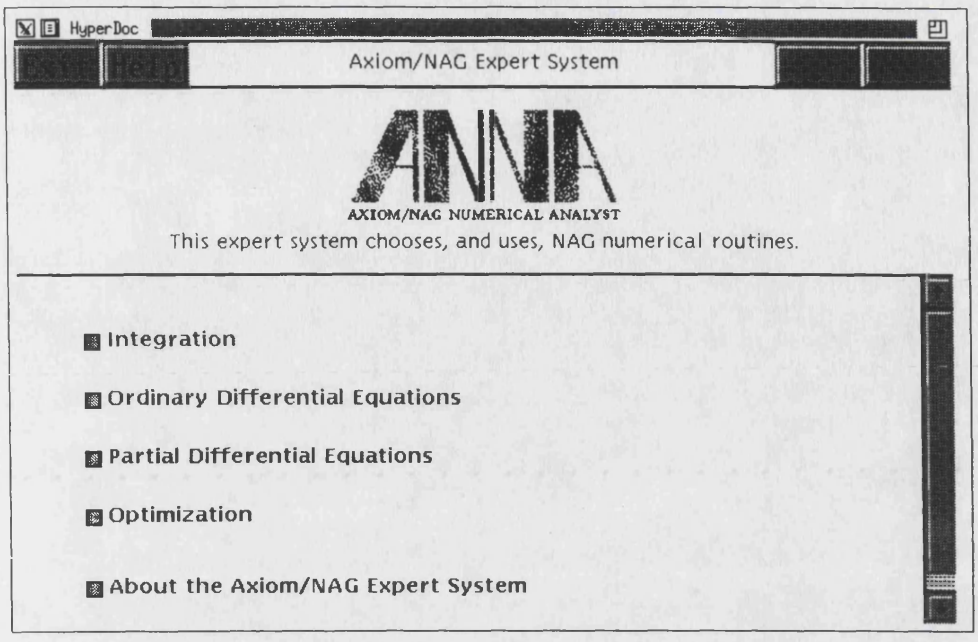


Figure 11-2: ANNA Top Level Page

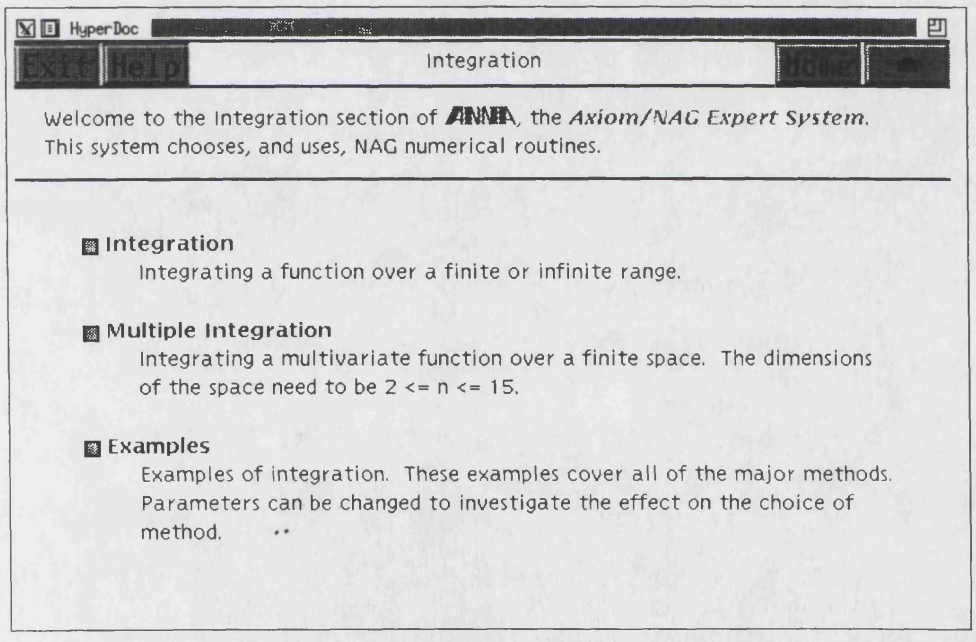


Figure 11-3: ANNA Integration Page

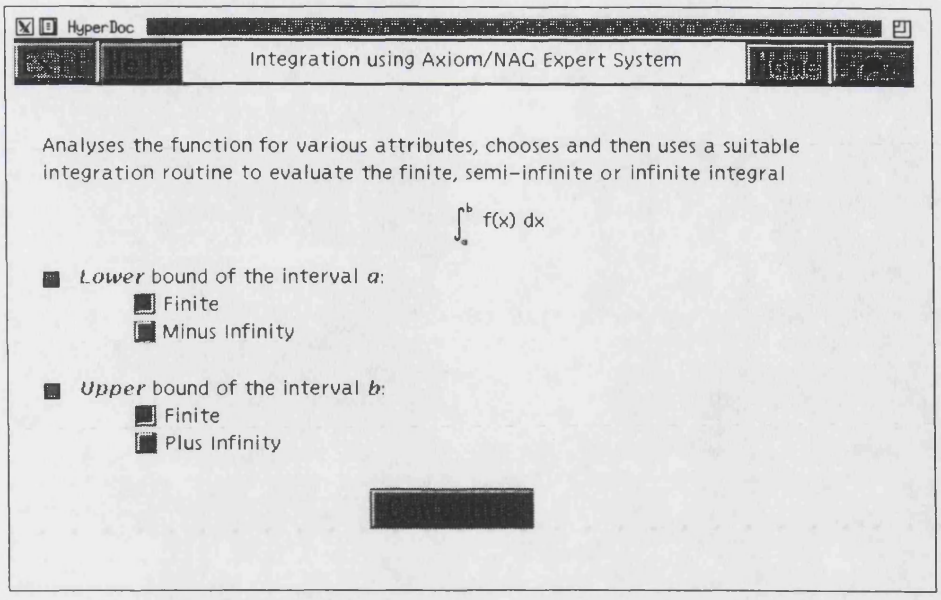


Figure 11-4: ANNA Integration Input Page 1

It is worth noting that, if the radio buttons for the range of integration in Figure 11-4 are differently selected, the page shown in Figure 11-5 would reflect those changes in that the default function given would be one defined on that range and the value `%plusInfinity` or `%minusInfinity` entered in the appropriate box as a reminder of the proper syntax.

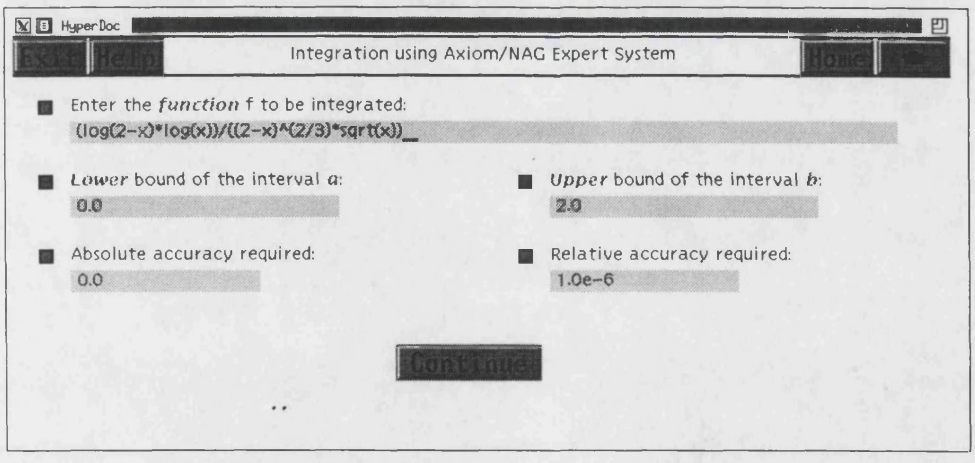


Figure 11-5: ANNA Integration Input Page 2

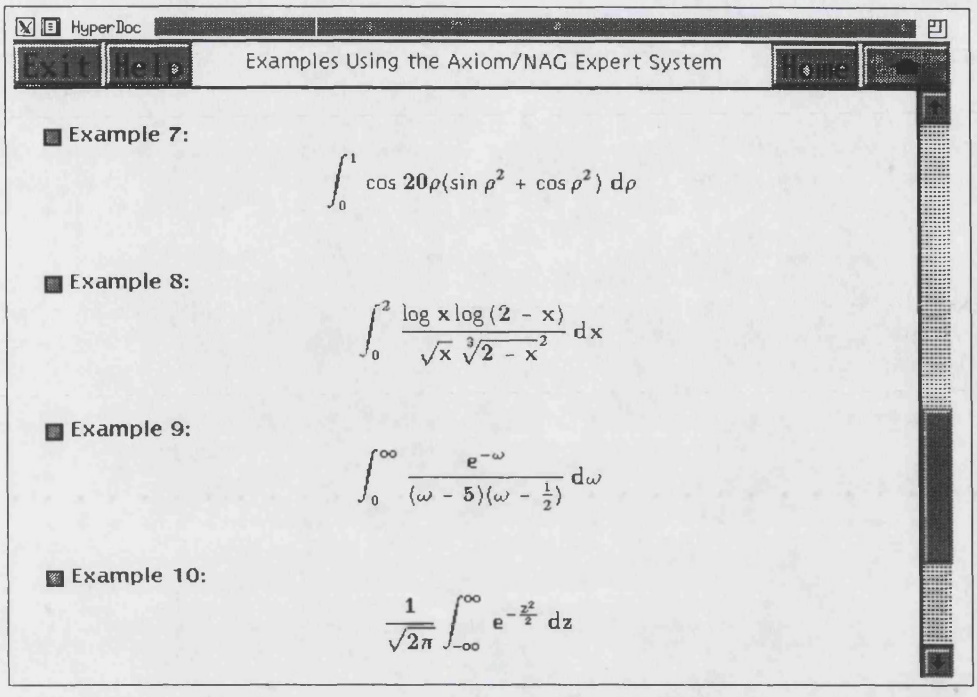


Figure 11-6: ANNA Integration Examples Page

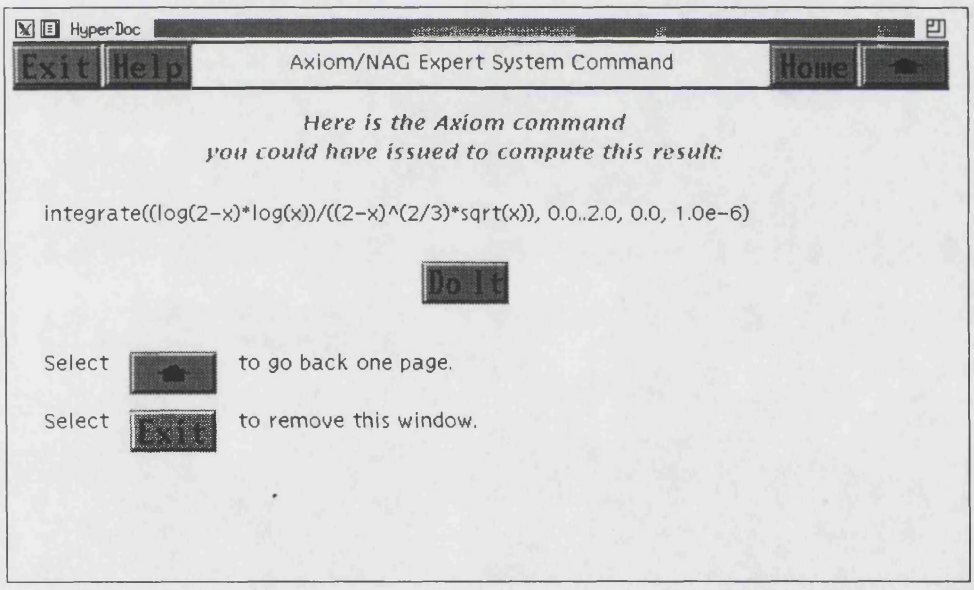


Figure 11-7: ANNA Integration Instigation Page

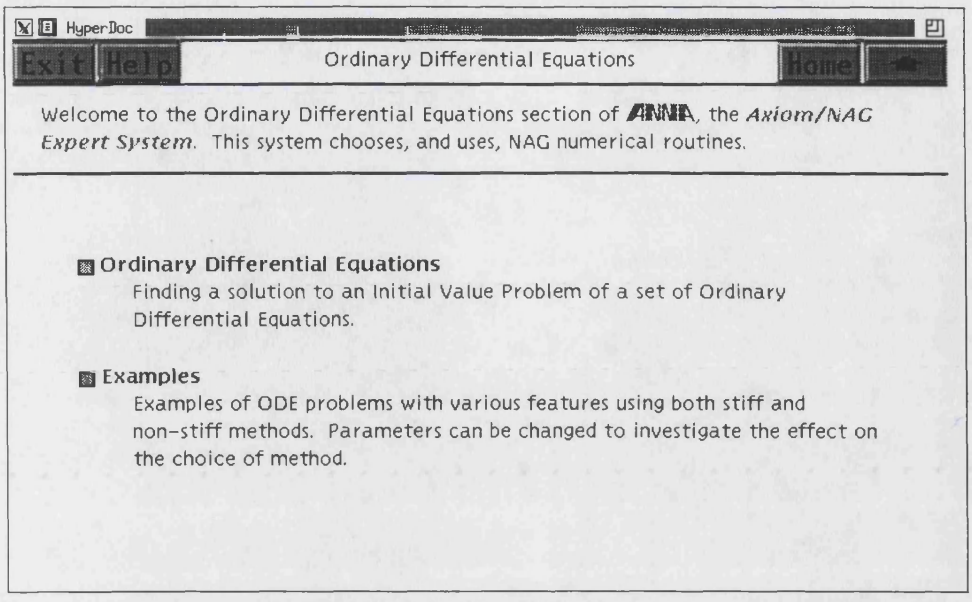


Figure 11-8: ANNA Ordinary Differential Equations Page

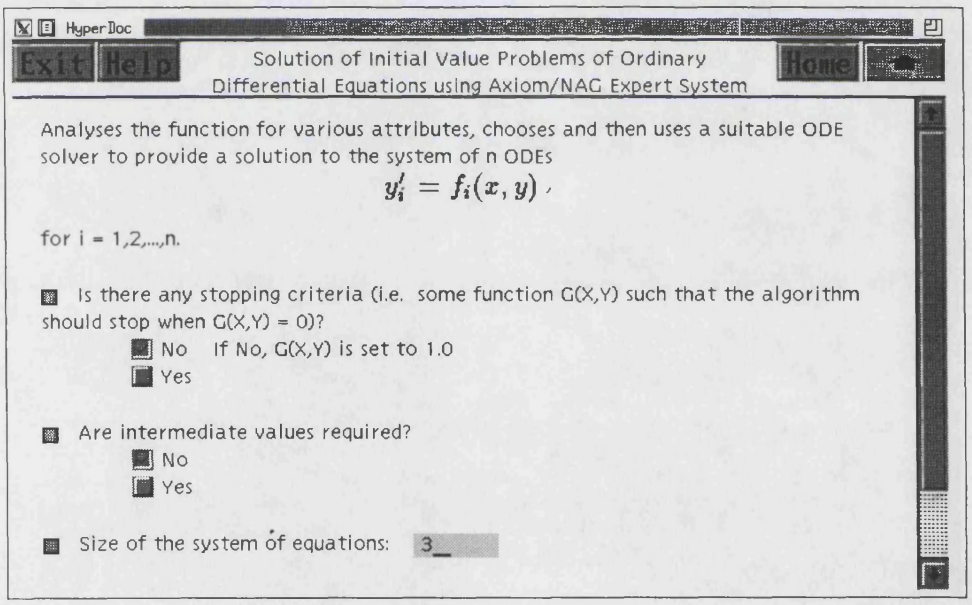


Figure 11-9: ANNA Ordinary Differential Equations Input Page 1

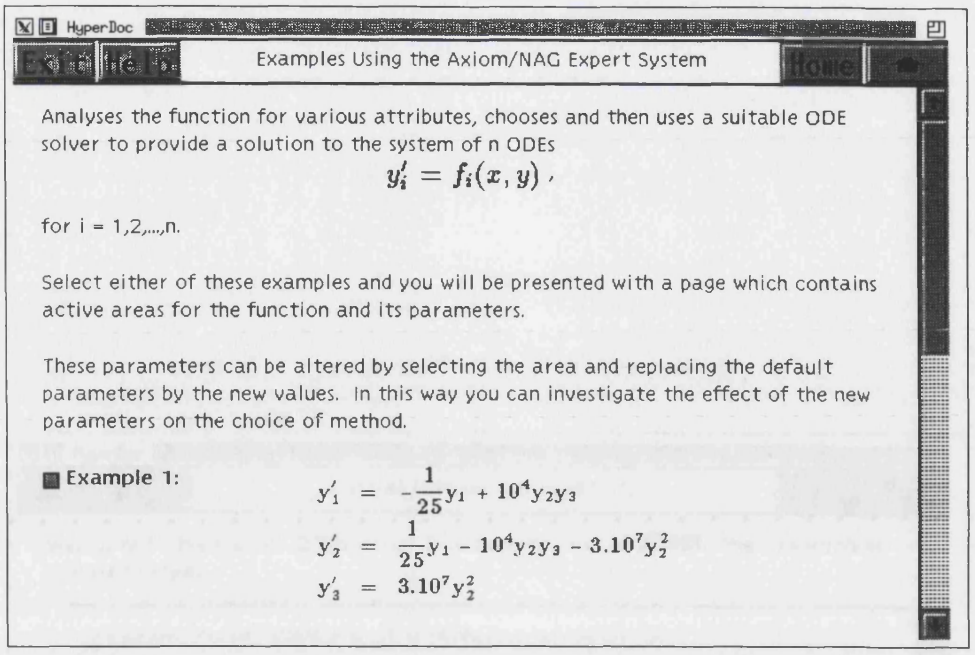


Figure 11-10: ANNA Ordinary Differential Equations Examples Page

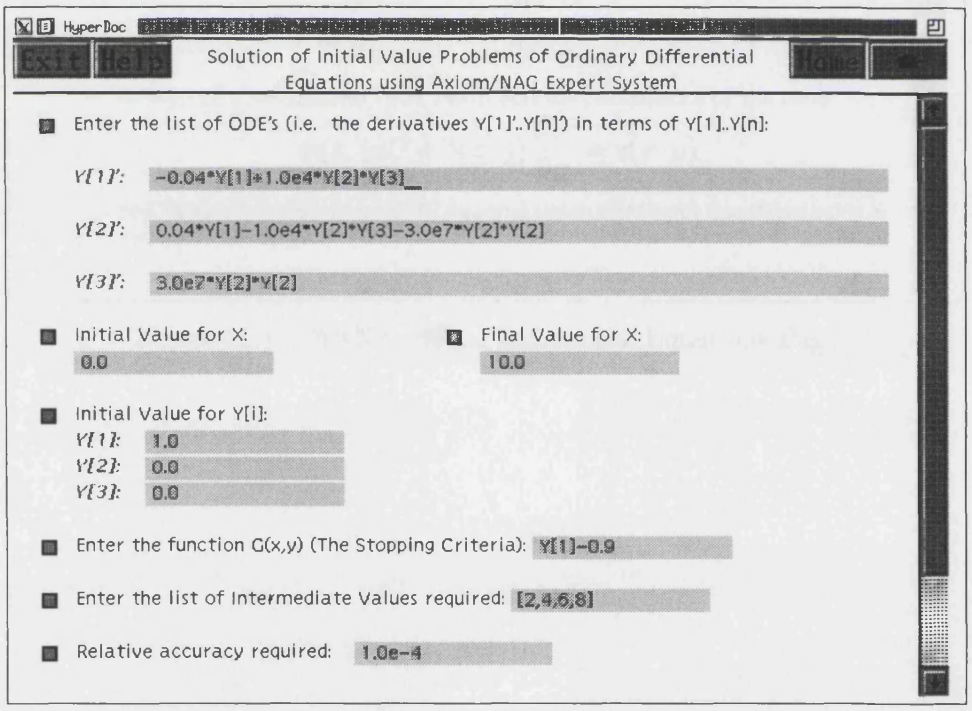


Figure 11-11: ANNA Ordinary Differential Equations Input Page 2

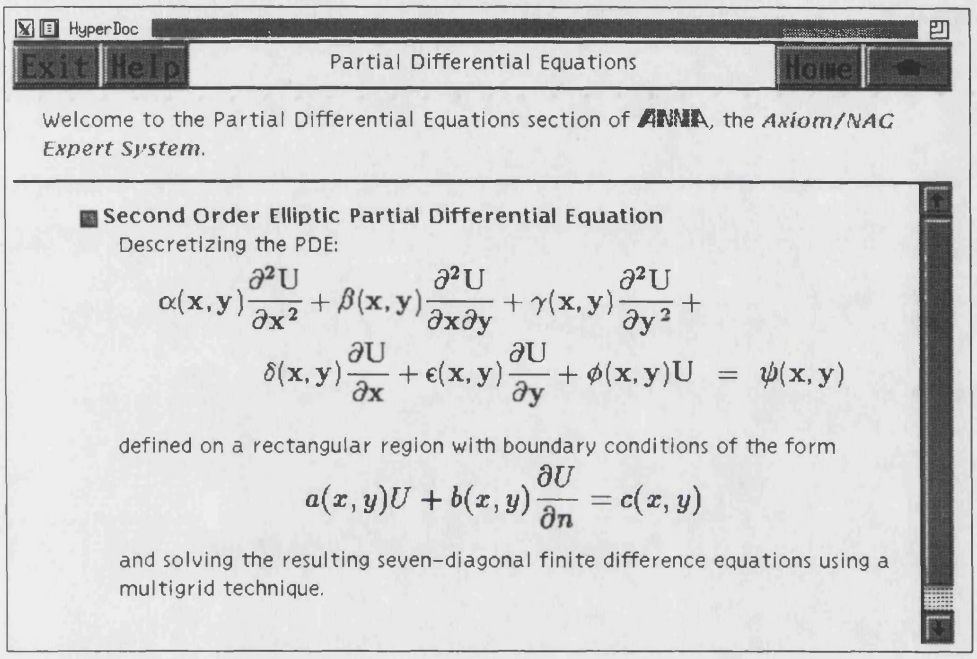


Figure 11-12: ANNA Partial Differential Equations Page

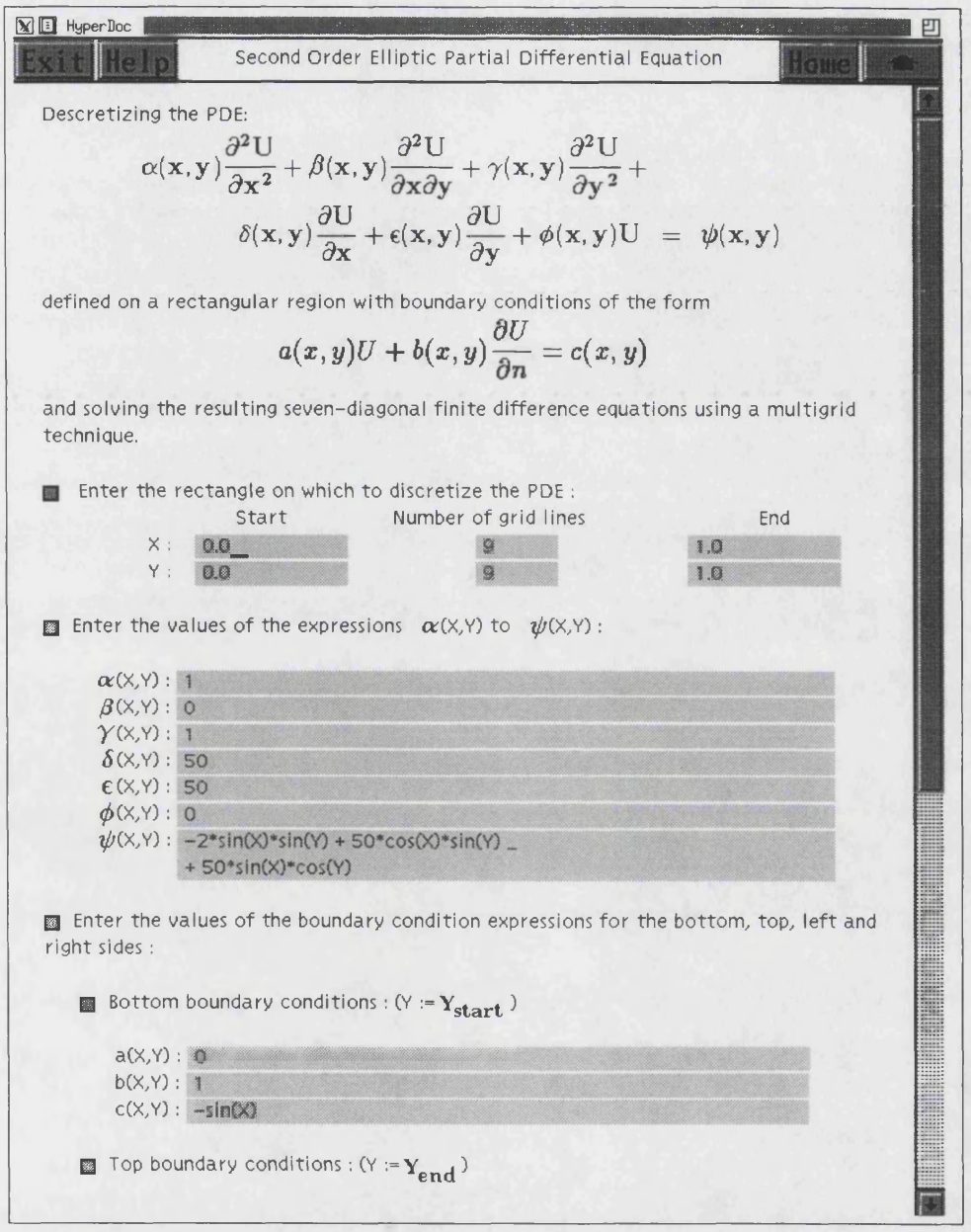


Figure 11-13: ANNA Partial Differential Equations Input Page

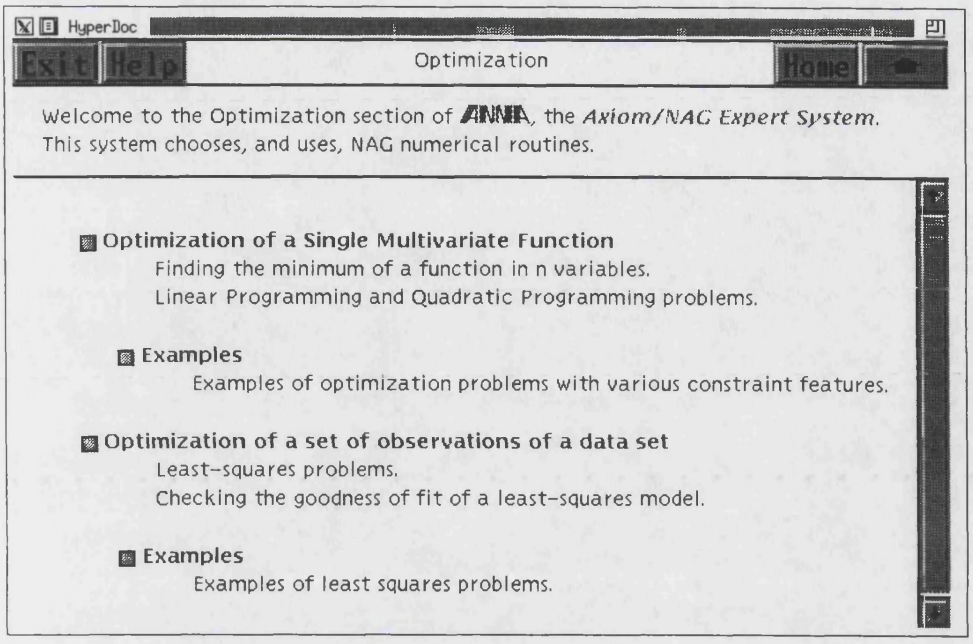


Figure 11-14: ANNA Optimization Page

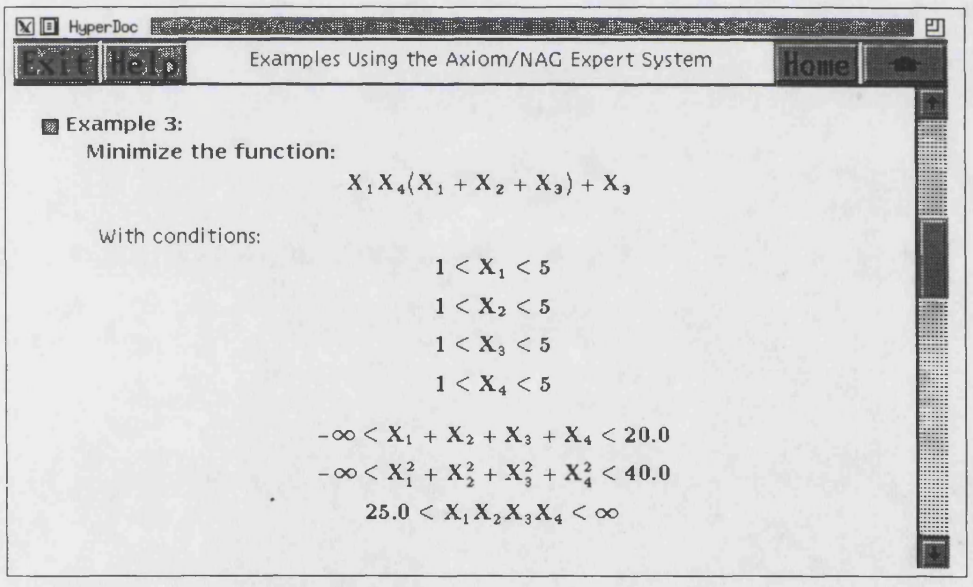


Figure 11-15: ANNA Optimization Examples Page

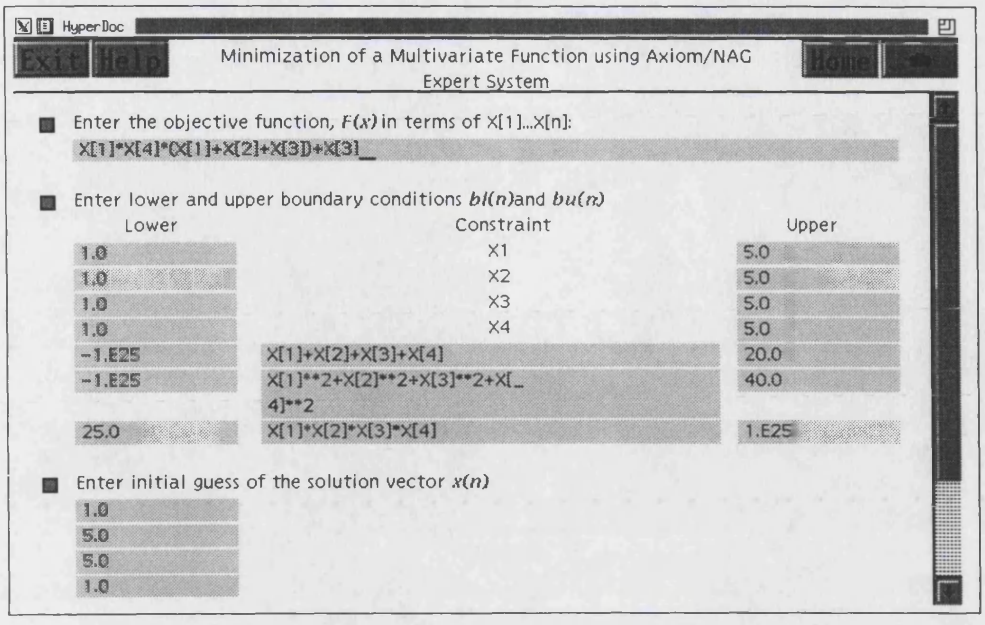


Figure 11-16: ANNA Optimization Input Page

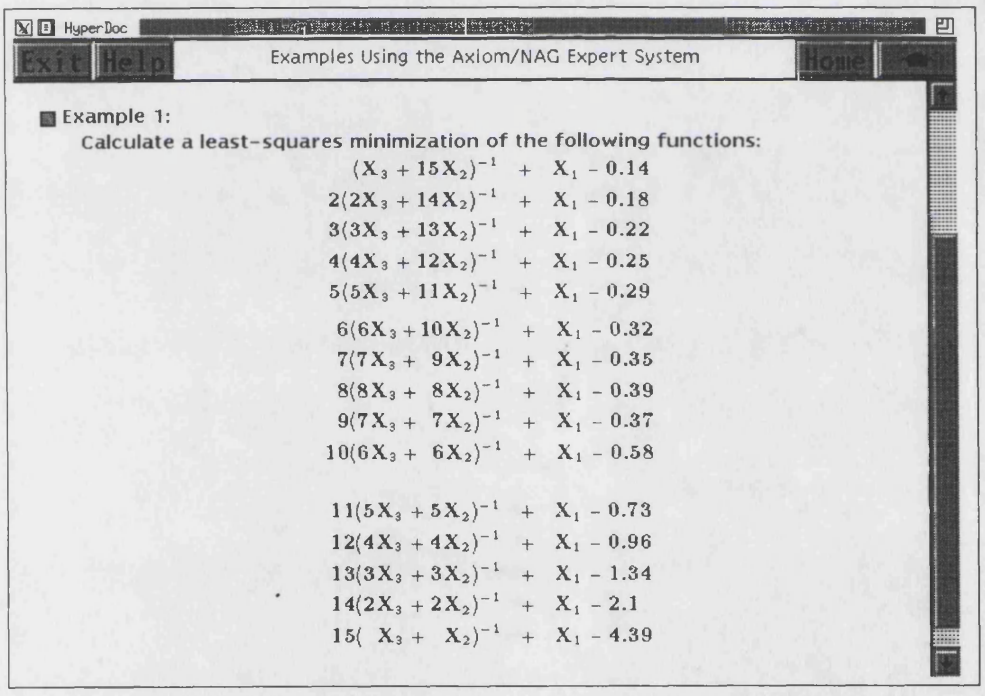


Figure 11-17: ANNA Least-Squares Optimization Examples Page

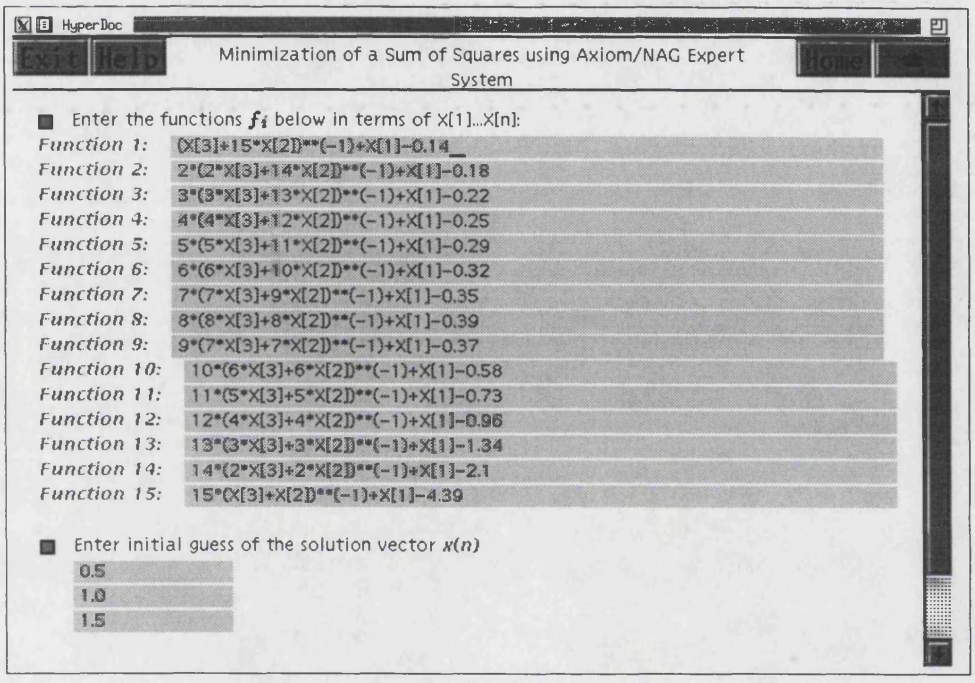


Figure 11-18: ANNA Least-Squares Optimization Input Page

Chapter 12

Conclusion

In summary, the background to the problem which is the subject of this thesis will be discussed with regards to the history of the use of numerical methods, the development of Computer Algebra Systems and the introduction of improved graphical interfaces. Some insight will be provided into the reasoning behind the decisions made regarding how a solution could be constructed as well as some of the tools necessary for its construction, such as expert system technology and the knowledge base, the extension of Dempster-Shafer theory and Lucks/Gladwell intensity, compatibility and aggregation functions to provide a sound framework for the decision process, the explanation mechanism which can inform the user about the reasoning and decision processes and the use of facilities provided within Axiom to investigate attributes which affect such a decision.

The proposed, and implemented, solution is described and evaluated, both in its ability to tackle the wide range of numerical problems and in the interface that it provides to numerical software. I will also discuss how it can be used in the construction of composite algorithms (those using both numerical and symbolic processes) and the further work currently being undertaken.

12.1 The Problem and its Background

The difficulty of using numerical methods is illustrated by the number of books and programs dedicated, in full or in part, to making the subject easier. However they cannot give advice on whether this or that routine or method can solve a given problem

and, since, in some problem domains, there are often a number of routines which might possibly be addressed to a particular problem, some of which might work, others may be inefficient and yet others may fail, this choice is important. Most of these books and programs do not even give advice on how to attempt to find out!

Admittedly, libraries might only provide a single routine for each problem type, but this is by no means always the case. In particular, in the areas of numerical integration, solution of initial value problems of ordinary differential equations and optimization, there are a number of routines for which the user is expected to select the routine most appropriate. This process may be fairly straightforward, but require extensive analysis skills, or it may be subtle, requiring a considerable understanding of both the problem and the possible numerical methods. Whatever problem the user is confronted with, the use of numerical library routines need programming ability — often in Fortran.

The growth in the use and capabilities of computer programs, and Computer Algebra Systems in particular, have shown that users wish to have a significantly easier and more friendly interface to computation in general. However, the power and reliability of numerical routines and numerical libraries should not be jettisoned just because of the difficulties inherent in the use of such programs.

What is required is a simpler and more intelligent interface — one that can decide, given any problem (within some domain), what are the attributes that affect the choice of routine and thus make a decision as to a likely contender for its solution before implementing such a routine or method. Most attempts to consider this problem in the past have been either incomplete or limited in scope, relying mainly on decision trees and the user's own knowledge of possibly complicated mathematical concepts.

Given that the particular implementation is to utilise the links newly available for the CAS Axiom to call NAG Numerical Library routines (specifically those provided within the Foundation Library, an important subset of the main Fortran Library), the interface should retain the feel of Axiom whilst still providing the large amount of output information normal for library routines which can thus be used and inspected at will. It would also need to act as a 'black box', as required, and be capable of making any analysis of the input data, make an appropriate choice of routine, make the call to that routine automatically and, before output of the result, satisfy itself that the result obtained is sufficient to the users requirements (assuming that a result is possible or makes sense), both in type and accuracy.

12.2 The Solution

There are a number of different requirements in a solution corresponding to the demands of the user and the individual problem domains. To provide an automatic analysis of the problem, a number of computational agents have been created, varying in size and complexity from a few lines of code, as in the agent for the range of integration, to the agent for stiffness of a set of ODEs which is far more extensive. These can themselves use a range of techniques to provide answers to each question. They may be deterministic or may use 'rule of thumb' or even probabilistic components as well as composite algorithms (those which include both symbolic and numerical processes).

For example, the continuity of an elementary function is not always determinable. The computational agent thus is required to use a range of techniques which will attempt to provide a list of 'possible' problem points (at least those which Fortran programs would consider difficult). It can use look-up tables, inverse functions as well as numerical methods such as Sturm sequences to identify as many problem points as possible.

A further example is the computational agent used to investigate the stiffness of a set of ordinary differential equations. This also uses a variety of techniques including evaluating the Jacobian symbolically and, if necessary, numerically using NAG Library routines.

The expert system, which uses these computational agents, was created using the object-oriented programming language supplied with Axiom. Using object-oriented techniques, the knowledge base was built as a number of Method Domains, each of which contained sufficient knowledge to identify how suitable it is to solve the current problem and how to implement that method. They also provide some information for the explanation process.

Within the process to assess the ability of each method to be addressed to a given problem, it uses a number of techniques from measure theory. In particular, an extension of Dempster-Shafer theory to include limited recursivity, which allows for a problem to be divided into sections and separate techniques to be applied to each part, and Lucks/Gladwell measure functions were employed. This could provide a fuller analysis where conflicting information is supplied by the computational agents.

Further Domains controlled the knowledge of the range of techniques and default values as well as stored knowledge to be used within the system and within the explanation mechanism.

All these constituent parts are brought together in a single unit which uses a graphical interface using web-based technology. This allows users unfamiliar with programming techniques to employ the power and reliability of numerical methods without the fuss of either writing Fortran programs or of the necessity of analysing each problem to identify the best choice of method.

12.3 Evaluation

The system has undergone considerable testing and evaluation. The test results are described in Appendix B.3. For each chapter, a number of examples were used and note was taken of details such as the name of the chosen routine given certain parameters. This was then compared with published recommendations [Enright *et al.*, 1975; Hull *et al.*, 1972]. Where appropriate, the times taken for the computation using the recommended method were compared to other methods.

For the integration chapter, the tests did not cover all combinations of all routines, but concentrated on the ability of the expert system to correctly identify certain characteristics of the integrand and range and so make a good general choice of routine. Tests also covered the ability to use fall-back strategies.

The tests for the chapter on Ordinary Differential Equations used mainly published examples from [NAG, 1996; Enright *et al.*, 1975; Hull *et al.*, 1972] since these were readily available and comprehensive. Of the non-stiff problems, ANNA correctly chose only non-stiff methods and the chosen particular routines were in line with the recommendations in [Hull *et al.*, 1972].

The stiff problems from [Enright *et al.*, 1975] were tackled, in general, quite well. For those cases where the stiffness could be considered mild or very mild, a non-stiff method was preferred. This was confirmed by considering the timings using the specific methods directly instead of using ANNA. In most cases, by increasing the sensitivity of ANNA to stiffness, the system could be forced into choosing a stiff routine. Conversely, increasing the required accuracy (or increasing the sensitivity to accuracy or complexity) had the opposite effect.

In one case, the stiffness detection algorithm failed to notice that the problem exhibited increasing stiffness over the range of integration and thus chose a non-stiff routine where, if the range of integration was large, a stiff solver would be better. There was one other case where ANNA correctly identified that a system was very stiff, but still,

erroneously, chose a non-stiff routine. Further work would be needed to understand the cause of this failure. All tests based on examples provided by [NAG, 1996] were correctly analysed.

The optimization chapter was tested mainly by using examples in [NAG, 1996]. This again concentrated on the correct identification of characteristics of the input problem i.e. the type of objective function and constraints (linear, quadratic etc.) as well as continuity.

In general, the number of input parameters to ANNA is significantly less than that required for the direct use of specific routines, since it includes the ability to calculate many of the required extra parameters. As a consequence, since the input parser of Axiom spends considerable time identifying and categorising input data[1], whereby the calculation of these parameters can be very efficient, ANNA can be shown to be more efficient than using the link to the individual NAG routines directly. This is particularly apparent when one considers, say, the routine `E04UCF` which requires 43 parameters whereas to call the same routine using ANNA requires only 6.

12.4 Summary and Further Work

- I have created an expert system using the Axiom Symbolic Algebra System which can, reliably and automatically, decide which numerical routine is best or could be best for a given integration, ODE or optimization problem.

  Whilst this has been attempted before (usually only for a single problem domain), previous implementations have either required user input to answer possibly complex mathematical questions or they are severely restricted in their scope or reliability.

  Furthermore, I have shown that such an expert system can be created using Axiom's own object-oriented programming language. This expert system contains all the required components of an identifiable discrete knowledge base, inference mechanism and explanation process (p. 33) as well as fulfilling the [Newell & Simon, 1972] requirements of an Information Processing System.

- I have integrated this with the link to library routines which therefore automatically perform the required calculation.

[1] Axiom, being object-oriented with the inherent strong typing, requires that any input data be analysed by the parser and allocated a type. This can be a fairly expensive process. The calculation of these parameters by ANNA is quicker since the types are defined by the program.

Again, previous systems have (except for IRENA/ARC) only partially implemented this section either by automatically writing the Fortran code for later implementation or simply directed the user to the particular routine.

- I have provided a user interface to the expert system which is both easy to use and is consistent with the standard Axiom user interface and syntax.

  Systems in use by other Computer Algebra packages e.g. Maple, provide a very basic interface to numerical methods without providing the expert system.

- I have shown how measure theory can be used to provide a sound theoretical basis for decision making where multiple strategies and conflicting evidence is present.

  This has required an extension to Dempster-Shafer theory to cover limited recursivity together with Lucks/Gladwell measure functions.

- I have shown how composite techniques i.e. techniques using both numerical and symbolic parts, are used within the system and can be used interactively to increase the applicability of both computer algebra systems and numerical libraries

It is apparent that ANNA makes data input for numerical routines much easier. However, it does not tackle the problem of putting the numerical output in an easier to understand form. It would therefore be useful to create routines which could easily display the numerical results graphically. For example, the intermediate results of an ODE calculation could be plotted, together with some interpolation on these results if necessary, which would give further insight to the problem in hand.

Further work could also include the use of ANNA, and numerical routines in general, as part of algorithms containing both numerical and symbolic processes. This leads to a much greater use of the technological capabilities and a number of interesting possibilities, particularly in the field of ordinary and partial differential equation solvers. These now algorithms could then be incorporated within ANNA and used within further research.

Given the possibility of links to other libraries or additions to the NAG Foundation Library, further routines and methods could also be added into ANNA. This would only entail providing an Axiom Method Domain for that routine and updating the database of methods, plus, maybe, any computational agents to provide the analysis necessary to distinguish between the possible strategies.

References

BACKUS, J. 1981. The History of Fortran I, II, and III. *In:* [Wexelblat, 1981].

BIRKHOFF, G., & ROTA, G-C. 1978. *Ordinary Differential Equations.* 3rd edn. New York: John Wiley & Sons.

BOISVERT, R. F. 1989. The Guide to Available Mathematical Software Advisory System. *Mathematics and Computers in Simulation*, 453–463. Also published in [Houstis *et al.*, 1990, 167–178].

BOOCH, G. 1994. *Object-Oriented Analysis and Design with Applications.* 2nd edn. Redwood City, California: Benjamin/Cummings.

BROUGHAN, K. A., KEADY, G., ROBB, T., RICHARDSON, M. G., & DEWAR, M. C. 1991. Some symbolic computing links to the NAG numeric library. *SIGSAM Bulletin*, **25**(July 1991), 28–37.

BROWNSTON, L., FARRELL, R., KANT, E., & MARTIN, N. 1985. *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming.* Reading, MA: Addison-Wesley.

BUCHANEN, B. G., & SHORTLIFFE, E. H. (eds). 1984. *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Reading, Mass.: Addison Wesley.

BUCHBERGER, B., COLLINS, G. E., & LOOS, R. WITH ALBRECHT, R. (eds). 1983. *Computer Algebra: Symbolic and Algebraic Computation.* Wien: Springer-Verlag.

CARDELLI, L., & WEGNER, P. 1985. On understanding Types, Data Abstraction and Polymorphism. *Computing Surveys*, **17**(4).

CARHART, R. E. 1979. CONGEN: An Expert System Aiding the Structural Chemist. *In:* [Michie, 1979].

CHENEY, W., & KINCAID, D. 1985. *Numerical Mathematics and Computing.* 2nd edn. London: Chapman & Hall.

COLLINS, G. E., & LOOS, R. 1983. Real Zeros of Polynomials. *In:* [Buchberger *et al.*, 1983].

CONTE, S. D., & DE BOOR, C. 1980. *Elementary Numerical Analysis: An Algorithmic Approach.* New York: McGraw-Hill.

DAHL, O-J., DIJKSTRA, E. W., & HOARE, C. A. W. 1972. *Structured Programming.* London: Academic Press.

DAVENPORT, J. H. 1981. *On the Integration of Algebraic Functions.* Lecture Notes in Computer Science, vol. 102. Berlin: Springer-Verlag.

DAVENPORT, J. H. 1982. On the Parallel Risch Algorithm (I). *Pages 144–157 of: EUROCAM '82.* Lecture Notes in Computer Science, vol. 144. Marseilles: Springer Verlag, Berlin.

DAVENPORT, J. H., SIRET, Y., & TOURNIER, E. 1988. *Computer Algebra: Systems and Algorithms for Algebraic Computation.* London: Academic Press.

DAVENPORT, J. H., DEWAR, M. C., & RICHARDSON, M. G. 1991. Symbolic and Numeric Computation: The IRENA Project. *Pages 1–18 of: Workshop on Symbolic and Numeric Computation.*

DAVIS, R., & LENAT, D. B. 1982. *Knowledge-Based Systems in Artificial Intelligence.* New York: McGraw-Hill.

DEKKER, K., & VERWER, J. D. 1984. *Stability of Runge-Kutte Methods for Stiff Nonlinear Differential Equations.* Amsterdam: North Holland.

DEKKER, T. J. 1980. Design of Languages for Numerical Algorithms. *In:* [Hennell & Delves, 1980].

DEWAR, M. C. 1991. *Interfacing Algebraic and Numeric Computation.* Ph.D. thesis, University of Bath.

DEWAR, M. C. 1992. Using Computer Algebra to Select Numerical Algorithms. *Pages 1–8 of:* WANG, P.S. (ed), *ISSAC 1992.* Berkeley, Calif.: ACM, New York.

DU CROZ, J. J. 1982. Programming Languages for Numerical Subroutine Libraries. *In:* [Reid, 1982].

DUPÉE, B. J. 1996. Measuring the Likely Effectiveness of Strategies. *In:* CALMET, J. CAMPBELL, J.A., & PFALZGRAF, J. (eds), *AISMC-3: Artificial Intelligence for Symbolic and Mathematical Computation.* Lecture Notes in Computer Science, vol. 1138. Steyr, Austria: Springer Verlag, Berlin.

DUPÉE, B. J., & DAVENPORT, J. H. 1995. Using Computer Algebra to Choose and Apply Numerical Routines. *AXIS*, **2**(3), 31–41.

DUPÉE, B. J., & DAVENPORT, J. H. 1996. An Intelligent Interface to Numerical Routines. *Pages 252–262 of:* CALMET, J., & LIMONGELLI, J. (eds), *DISCO'96: Design and Implementation of Symbolic Computation Systems.* Lecture Notes in Computer Science, vol. 1128. Karlsrühe: Springer Verlag, Berlin.

ENRIGHT, W. H., HULL, T. E., & LINDBERG, B. 1975. Comparing Numerical Methods for Stiff Systems of O.D.E.'s. *BIT*, **15**, 10–48.

FORD, B., HAGUE, S. J., & ILES, R. M. J. 1989. Numerical Knowledge-Based Systems. *Mathematics and Computers in Simulation*, 395–400. Also published in [Houstis *et al.*, 1990, 281–286].

FROST, R. A. 1986. *Introduction to Knowledge Base Systems.* London: Collins.

GAFFNEY, P. W., WOOTEN, J. W., KESSEL, K. A., & MCKINNEY, W. R. 1983. NITPACK: An Interactive Tree Package. *ACM Transactions on Mathematical Software*, **9**(4), 395–417.

GEAR, C. W. 1971. *Numerical Initial-Value Problems in Ordinary Differential Equations.* Englewood Cliffs, NJ: Prentice-Hall.

GEDDES, K. O., CZAPOR, S. R., & LABAHN, G. 1992. *Algorithms for Computer Algebra.* Boston, Mass.: Kluwer Academic Publishers.

GLADWELL, I., & LUCKS, M. 1992. An Interactive Session with a Knowledge Based System for Mathematical Software Selection. *In:* [Houstis & Rice, 1992].

GORDON, J., & SHORTLIFFE, E. H. 1983. The Dempster-Shafer Theory of Evidence and its Relevance to Expert Systems. *In:* [Buchanen & Shortliffe, 1984].

GORDON, J., & SHORTLIFFE, E. H. 1985. A Method of Managing Evidential Reasoning in a Hierarchical Hypothesis Space. *Artificial Intelligence*, **26**(3), 323–357.

HALL, G., & WATT, J. M. (eds). 1976. *Modern Methods for Ordinary Differential Equations.* Oxford: Clarendon Press.

HAWKES, E., & KEADY, G. 1995. Two more links to NAG numerics involving CA systems. *In: IMACS Conference on Applications of Computer Algebra, May 16-19, 1995.*

HAZEL, P., & O'DONOHOE, M. R. 1980. HELP Numerical: The Cambridge Interactive Documentation System for Numerical Methods. *In:* [Hennell & Delves, 1980].

HECK, A. 1996. *Introduction to Maple.* 2nd edn. New York: Springer Verlag.

HENNELL, M. A., & DELVES, L. M. (eds). 1980. *Production and Assessment of Numerical Software.* London: Academic Press. Based on the Proceedings of the Conference on the Production and Assessment of Numerical Software (NS79), Liverpool, April 1979.

HINDMARSH, A. C. 1983. ODEPACK: A Systemized Collection of ODE Solvers. *In:* [Stepleman, 1983].

HOPKINS, T., & PHILLIPS, C. 1988. *Numerical Methods in Practice: Using the NAG Library.* Wokingham, England: Addison-Wesley.

HOUNHAM, I. 1996. *Calling NAG Fortran Library Routines from C Language Programs Using the NAG C Header Files.* Tech. rept. NAG Ltd, Oxford. NAG Publication Code NP2007.

HOUSTIS, E. N., & RICE, J. R. (eds). 1992. *Artificial Intelligence, Expert Systems and Symbolic Computing.* Amsterdam: North-Holland. Based on the Proceedings of the 13th World Congress on Computation and Applied Mathematics, Dublin, July 1991.

HOUSTIS, E. N., RICE, J. R., & VICHNEVETSKY, R. (eds). 1990. *Intelligent Mathematical Software Systems.* Amsterdam: North-Holland. Proceedings of the First IMACS International Conference on Expert Systems for Numerical Computing, Purdue University, April 1988.

HOUSTIS, E. N., RICE, J. R., & VICHNEVETSKY, R. (eds). 1992. *Expert Systems for Scientific Computing.* Amsterdam: North-Holland. Proceedings of the Second IMACS International Conference on Expert Systems for Numerical Computing, Purdue University, April 1992.

HULL, T. E., ENRIGHT, W. H., FELLEN, B. M., & SEDGWICK, A. E. 1972. Comparing Numerical Methods for Ordinary Differential Equations. *SIAM J. Numer. Anal.*, **9**(4), 603–637.

JACKSON, P. 1992. *Introduction to Expert Systems.* 2nd edn. Reading, MA: Addison-Wesley.

JACOBS, D. A. H. (ed). 1978. *Numerical Software — Needs and Availability.* London: Academic Press. Proceedings of the Conference on Applications of Numerical Software — Needs and Availability, University of Sussex, Sept 1977.

JENKS, R. D., & SUTOR, R. S. 1992. *AXIOM: The Scientific Computation System.* New York: Springer-Verlag.

KAMEL, M. S., MA, K. S., & ENRIGHT, W. H. 1993. ODEXPERT: An Expert System to Select Numerical Solvers for Initial Value ODE Systems. *ACM Transactions on Mathematical Software*, **19**(1), 44–62. Also published in [Houstis *et al.*, 1992, 33–54].

KEMP, P. 1978. Libraries: The User Interface. *In:* [Jacobs, 1978].

KNUTH, D. E. 1981. *The Art of Computer Programming, Volume 2: Seminumerical Algorithms.* 2nd edn. Reading, MA: Addison Wesley.

LAMBERT, J. D. 1973. *Computational Methods in Ordinary Differential Equations.* London: Wiley.

LUCKS, M., & GLADWELL, I. 1992. Automated Selection of Mathematical Software. *ACM Transactions on Mathematical Software*, **18**(1), 11–34. Also published in [Houstis *et al.*, 1992, 421–459] as 'A Functional Representation for Software Selection Expertise'.

MICHIE, D. (ed). 1979. *Expert Systems in the Micro-Electronic Age.* Edinburgh: Edinburgh University Press. Based on the proceedings of the 1979 Artificial Intelligence and Simulation of Behaviour Summer School on Expert Systems.

NAG. 1989. *The KASTLE System.* Tech. rept. The FOCUS Consortium, NAG Ltd, Oxford. NAG Publication Code NP2022.

NAG. 1996. *Fortran Library Manual - Mark 17.* NAG Ltd, Oxford, UK. NAG Publication Code NP2834.

NEWELL, A., & SIMON, H. A. 1972. *Human Problem Solving.* Eaglewood Cliffs, N.J.: Prentice-Hall.

PARIS, J. B. 1994. *The Uncertain Reasoner's Companion: A Mathematical Perspective.* Cambridge, UK: CUP.

PEARL, J. 1986. On Evidential Reasoning in a Hierarchy of Hypotheses. *Artificial Intelligence*, **28**(1), 9–15.

POSTEL, F., & ZIMMERMAN, P. 1996. A Review of the ODE Solvers of AXIOM, Derive, Macsyma, Maple, Mathematica, MuPAD and Reduce. *In: 5th Rhine Workshop on Computer Algebra, April 1-3, 1996.*

PRINGLE-PATTISON, A. S. (ed). 1969. *John Locke: An Essay Concerning Human Understanding.* Oxford, England: OUP.

PROTHERO, A. 1976. Introduction to Stiff Problems. *In:* [Hall & Watt, 1976].

REID, J. K. (ed). 1982. *The Relationship Between Numerical Computation and Programming Languages.* Amsterdam: North Holland. Proceedings of the IFIP TC2 Workshop, Boulder, Colorado, Aug 1981.

RICHARDSON, D. 1968. Some Undecidable Problems Involving Elementary Functions of a Real Variable. *J. Symbolic Logic*, **33**(4), 514–520.

RISCH, R. H. 1969. The Problem of Integration in Finite Terms. *Trans. ACM*, **139**, 167–189.

SCHULZE, K., & CRYER, C. W. 1983. NAXPERT: A Prototype Expert System for Numerical Software. *SIAM Journal of Scientific and Statistical Computing*, **9**(3), 503–515.

SHAFER, G. 1976. *A Mathematical Theory of Evidence.* Princeton, NJ: Princeton University Press.

SHAFER, G., & LOGAN, R. 1987. Implementing Dempster's Rule for Hierarchical Evidence. *Artificial Intelligence*, **33**, 271–298.

SHAMPINE, L. F., & GORDON, M. K. 1975. *Computer Solution of Ordinary Differential Equations.* San Francisco, CA: Freeman.

SIBREE, J. (ed). 1894. *G. W. F. Hegel: The Philosophy of History.* London: Bell.

STEPLEMAN, R. [ET AL.] (ed). 1983. *Scientific Computing.* IMACS transactions on scientific computation. Amsterdam: North-Holland. Proceedings of the 10th IMACS World Congress on Systems Simulation and Scientific Computation, Montreal, Canada, 8-13 August 1982.

SUNSOFT. 1995. *Fortran 77 4.0 User's Guide.* Sun Misrosystems Inc., Mountain View, Calif. Part Number 802-2997-10.

TRUSTED, J. 1981. *An Introduction to the Philosophy of Knowledge.* London: Macmillan.

WEINER, J. L. 1979. *The Structure of Natural Explanations: theory and application.* Tech. rept. SP - 4025. System Development Corporation.

WEXELBLAT, R. L. (ed). 1981. *History of Programming Languages.* New York: Academic Press. From the ACM SIGPLAN History of Programming Languages Conference, Los Angeles, Calif., June 1–3, 1978.

Appendix A

Worked Examples using ANNA

A.1 Computational Agents

```
Axiom Computer Algebra System (Release 2.1)
Solaris 2.5 for Sun SPARC [SunPro cc]
------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave AXIOM and return to shell.
------------------------------------------------------------------------

Value = "ANNA loaded successfully"
```

This message shows that Axiom has found and loaded the ANNA Categories and Domains.

```
(1) ->
(1) -> s := singularitiesOf(1/(x*cos(x)),[x],-%pi..%pi)$ESCONT

   (1)  [1.5707963267948966,- 1.5707963267948966,0.0]
                                                    Type: Stream DoubleFloat
```

Because some of these routines are not automatically available outside the expert system, we need to identify the Axiom Package to which they belong. In this case ESCONT is shorthand for ExpertSystemContinuityPackage.

```
(2) -> a:Record(var:Symbol, fn:Expression DoubleFloat, range:Segment
OrderedCompletion DoubleFloat, abserr:DoubleFloat, relerr:DoubleFloat);
                                                                Type: Void
(3) -> a := [x,cos(20*x)*exp(x)/((x-%pi)*x),0..%pi,0.0,0.0]
```

```
   (3)
                                  x
                      cos(20.0x)%e
   [var= x, fn= ------------------------, range= 0.0..3.1415926535897931,
                 2
                x  - 3.1415926535897931x
    abserr= 0.0, relerr= 0.0]

      Type: Record(var: Symbol,fn: Expression DoubleFloat,range: Segment
       OrderedCompletion DoubleFloat,abserr: DoubleFloat,relerr: DoubleFloat)

(4) -> exprHasAlgebraicWeight(a)$DO1WGTS

   (4)  [- 1.0,- 1.0]
                                              Type: Union(List DoubleFloat,...)
(5) -> exprHasWeightCosWXorSinWX(a)$DO1WGTS

   (5)  [op= cos,w= 20.0]
                    Type: Union(Record(op: BasicOperator,w: DoubleFloat),...)
```

A.2 Integration

$$\textbf{Problem}: \quad \int_0^\infty \frac{e^{-x^3} + e^{-3x^2}}{\sqrt{x}}\, dx$$

```
Axiom Computer Algebra System (Release 2.1)
Solaris 2.5 for Sun SPARC [SunPro cc]
------------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave AXIOM and return to shell.
------------------------------------------------------------------------------

Value = "ANNA loaded successfully"
(1) ->
(1) -> showScalarValues true;

                                                                   Type: Boolean
(2) -> integrate((exp(-x^3)+exp(-3*x^2))/sqrt(x), 0.0..%plusInfinity, 1.0e-6)
nagman:acknowledging request for d01apf
nagman:connection successful to dictum.maths.bath.ac.uk
nagman:receiving results from dictum.maths.bath.ac.uk

nagman:acknowledging request for d01ajf
nagman:connection successful to dictum.maths.bath.ac.uk
nagman:receiving results from dictum.maths.bath.ac.uk
```

The Naglink manager (nagman) displays messages showing the routine used and the connection to the NAG Daemon (nagd). In this case, two routines have been used and their results combined.

```
   (2)
   [
    abserr: 2.69960156338737e-08, result: 3.23287256251958,

    attributes: List(Any),

    method:
       [nameOfRoutine: "d01TransformFunctionType",

         other:
           [
             d01transformextra:
               List
                  Record
                     :(str,String)
```

```
                ,
                    :(fn,Expression(DoubleFloat))
                ,
                    :(range,Segment(OrderedCompletion(DoubleFloat)))
                ,
                    :(ext,Result)
            ]
        ,
       allMeasures: List(String), bestMeasure: 0.6086956521 7391304348]
    ,

    d01apfAnnaTypeAnswer:
      [iw: Matrix(Integer), abserr: 7.38412656787854e-14,
       w: Matrix(DoubleFloat), ifail: 0, result: 3.147059114838,

       method:
          [nameOfRoutine: "d01apfAnnaType",
           other: [d01apfextra: List(DoubleFloat)],
           allMeasures: List(String),
           bestMeasure: 0.7]
        ,
       attributes: List(Any)]
    ,

    d01ajfAnnaTypeAnswer:
      [iw: Matrix(Integer), abserr: 2.6995941792608e-08,
       w: Matrix(DoubleFloat), ifail: 0, result: 0.085813447681579,

       method:
          [nameOfRoutine: "d01ajfAnnaType", other: [],
           allMeasures: List(String), bestMeasure: 0.4]
        ,
       attributes: List(Any)]
    ]
                                                              Type: Result
(3) -> qelt(%.method,allMeasures)
```

The command, qelt(a,b), is used (due to a bug in the current version of Axiom) to identify the field b in the composite object a. The identifier % is used to signify the previous output object.

```
   (3)
   ["Trying One-dimensional infinite integration routines",

    "d01amfmeasure: 0.5 - d01amf is a reasonable choice if the integral
```

```
    is infinite or semi-infinite and d01transform cannot do better than
    using general routines"
     ,
    "d01asfmeasure: 0.0 - d01asf: A suitable weight has not been found",

    "d01transformmeasure: 0.609 - The recommendation is to transform the
    function and use d01apfAnnaType and d01ajfAnnaType"
     ]
                                                              Type: List String
(4) ->
```

Problem : $\displaystyle\int_0^2 \frac{\log(2-x)\ \log x}{(2-x)^{2/3}\ \sqrt{x}}\ dx$

```
Axiom Computer Algebra System (Release 2.1)
Solaris 2.5 for Sun SPARC [SunPro cc]
------------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave AXIOM and return to shell.
------------------------------------------------------------------------------

Value = "ANNA loaded successfully"
(1) ->
(1) -> showScalarValues true;

                                                                   Type: Boolean
(2) -> integrate((log(2-x)*log(x))/((2-x)^(2/3)*sqrt(x)), 0.0..2.0, 1.0e-6)
nagman:acknowledging request for d01apf
nagman:connection successful to dictum.maths.bath.ac.uk
nagman:receiving results from dictum.maths.bath.ac.uk


   (2)
   [iw: Matrix(Integer), abserr: 2.47574139246241e-11, w: Matrix(DoubleFloat),
    ifail: 0, result: - 5.89188342020428,

    method:
       [nameOfRoutine: "d01apfAnnaType",
        other: [d01apfextra: List(DoubleFloat)], allMeasures: List(String),
        bestMeasure: 0.7]
     ,
    attributes: List(Any)]
                                                                   Type: Result
(3) -> qelt(%.method,allMeasures)

   (3)
   ["Trying One-dimensional finite integration routines",
    "d01aqfmeasure: 0.0 - d01aqf: A suitable weight function has not been
     found",
    "d01anfmeasure: 0.0 - d01anf: A suitable weight has not been found",
    "d01ajfmeasure: 0.4 - The general routine d01ajf is our default",
    "d01akfmeasure: 0.0 - d01akf: The expression shows little or no
     oscillation",
```

```
    "d01apfmeasure: 0.7 - Recommended is d01apf with c = -0.5,
     d = -0.6666666666 6666662966 and l = 4",
    "d01alf is no better than other routines"]
                                                          Type: List String
(4) -> %%(2).attributes

   (4)  [There are singularities at both end points,The range is finite,[]]
                                                        Type: List Any
```

A.3 Ordinary Differential Equations

Problem :

$$
\begin{aligned}
y_1' &= \tan y_3 \\
y_2' &= -0.032\frac{\tan y_3}{y_2} - 0.02\frac{y_2}{\cos y_3} \\
y_3' &= -\frac{0.032}{y_2^2}
\end{aligned}
$$

with initial conditions:

$$
\begin{aligned}
y_1(0) &= 0.5 \\
y_2(0) &= 0.5 \\
y_3(0) &= 0.2\pi
\end{aligned}
$$

```
Axiom Computer Algebra System (Release 2.1)
Solaris 2.5 for Sun SPARC [SunPro cc]
------------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave AXIOM and return to shell.
------------------------------------------------------------------------------

Value = "ANNA loaded successfully"
(1) ->
(1) -> showScalarValues true;

                                                                  Type: Boolean
(2) ->  solve([tan(Y[3]), -0.032*tan(Y[3])/Y[2]-0.02*Y[2]/cos(Y[3]),
-0.032/(Y[2]**2)], 0.0, 10.0, [0.5, 0.5, %pi*0.2], 1.0e-4)
nagman:acknowledging request for f02aff
nagman:connection successful to dictum.maths.bath.ac.uk
nagman:receiving results from dictum.maths.bath.ac.uk
```

The above nagman call relates to the eigenvalue code used in the computational agent which measures the stiffness and stability of the system of ODEs.

```
nagman:acknowledging request for d02bbf
nagman:connection successful to dictum.maths.bath.ac.uk
nagman:receiving results from dictum.maths.bath.ac.uk


   (2)
   [ifail: 0, intensityFunctions: List(String), tol: 0.0002,
```

```
    result: Matrix(DoubleFloat), y: Matrix(DoubleFloat),

    method:
       [nameOfRoutine: "d02bbfAnnaType", allMeasures: List(String),
        bestMeasure: 0.7077983781 2664880201]
     ,
    x: 10.0]
                                                              Type: Result

(3) -> qelt(%.method,allMeasures)

   (3)
   ["d02ejfmeasure: 0.282 - BDF method for Stiff Systems",
    "d02bbfmeasure: 0.708 - Runge-Kutta Merson method",
    "d02bhf is no better than other routines",
    "d02cjf is no better than other routines"]
                                                        Type: List String
(4) -> %%(2).intensityFunctions
```

The identifier %%(2) refers to the second output in the Axiom buffer.

```
   (4)
   ["stiffness: 0.0", "stability: 0.695", "expense: 0.114", "accuracy: 0.234",
    "intermediateResults: 0.0"]
                                                        Type: List String

(5) -> %%(2).y

   (5)  [- 3.62767857069111  0.633235902637208  - 1.05149500854554]
                                                  Type: Matrix DoubleFloat
```

Problem : Solve

$$\begin{aligned} y_1' &= -\frac{1}{25}y_1 + 10^4 y_2 y_3 \\ y_2' &= \frac{1}{25}y_1 - 10^4 y_2 y_3 - 3 \times 10^7 y_2^2 \\ y_3' &= 3 \times 10^7 y_2^2 \end{aligned}$$

with initial conditions:

$$\begin{aligned} y_1(0) &= 1 \\ y_2(0) &= 0 \\ y_3(0) &= 0 \end{aligned}$$

and stopping when: $y_1 = 0.9$

```
Axiom Computer Algebra System (Release 2.1)
Solaris 2.5 for Sun SPARC [SunPro cc]
------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave AXIOM and return to shell.
------------------------------------------------------------------------

Value = "ANNA loaded successfully"
(1) ->
(1) -> showScalarValues true;

                                                              Type: Boolean
(2) -> solve([-0.04*Y[1]+1.0e4*Y[2]*Y[3], 0.04*Y[1]-1.0e4*Y[2]*Y[3]
-3.0e7*Y[2]*Y[2], 3.0e7*Y[2]*Y[2]], 0.0, 10.0, [1.0, 0.0, 0.0], Y[1]-0.9,
[2,4,6,8], 1.0e-4)
nagman:acknowledging request for d02ejf
nagman:connection successful to dictum.maths.bath.ac.uk
nagman:receiving results from dictum.maths.bath.ac.uk

   (2)
   [ifail: 0, intensityFunctions: List(String), tol: 0.0001,
    result: Matrix(DoubleFloat), y: Matrix(DoubleFloat),

    method:
       [nameOfRoutine: "d02ejfAnnaType", allMeasures: List(String),
        bestMeasure: 0.4466108327 646621749]
     ,
    x: 4.37671333535702]
                                                              Type: Result
```

```
(3) -> qelt(%.method,allMeasures)

   (3)
   ["d02ejfmeasure: 0.447 - BDF method for Stiff Systems",
    "d02bbfmeasure: 0.009 - Runge-Kutta Merson method",
    "d02bhfmeasure: 0.008 - Runge-Kutta Merson method",
    "d02cjfmeasure: 0.168 - Adams method"]
                                                    Type: List String
(4) -> %%(2).intensityFunctions

   (4)
   ["stiffness: 1.0", "stability: 1.0", "expense: 0.127", "accuracy: 0.234",
    "intermediateResults: 0.077"]
                                                    Type: List String
(5) -> %%(2).y

   (5)  [0.9  2.17778054421343e-05  0.0999782221945582]
                                                Type: Matrix DoubleFloat
```

A.4 Partial Differential Equations

Problem: **Solve**

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} + 50\frac{\partial U}{\partial x} + 50\frac{\partial U}{\partial y} = -2\sin x \sin y + 50\cos x \sin y + 50 \sin x \cos y$$

with boundary conditions

$$\begin{aligned} x &:= 0 & \frac{\partial U}{\partial n} &= -\sin x \\ x &:= 1 & U &= \sin x \sin y \\ y &:= 0 & \frac{\partial U}{\partial n} &= -\sin y \\ y &:= 1 & U &= \sin x \sin y \end{aligned}$$

```
Axiom Computer Algebra System (Release 2.1)
Solaris 2.5 for Sun SPARC [SunPro cc]
------------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave AXIOM and return to shell.
------------------------------------------------------------------------------

Value = "ANNA loaded successfully"
(1) ->
(1) -> showScalarValues true;

                                                                   Type: Boolean
(2) -> solve(0.0, 1.0, 0.0, 1.0, 9, 9, [1, 0, 1, 50, 50, 0, -2*sin(X)*sin(Y) +
50*cos(X)*sin(Y) + 50*sin(X)*cos(Y)], [[ 0, 1, -sin(X)],[ 1, 0, sin(X)*sin(Y)]
,[ 1, 0, sin(X)*sin(Y)],[ 0, 1, -sin(Y)]], "elliptic", 1.0e-4 )
nagman:acknowledging request for d03eef
nagman:connection successful to dictum.maths.bath.ac.uk
nagman:receiving results from dictum.maths.bath.ac.uk

nagman:acknowledging request for d03edf
nagman:connection successful to dictum.maths.bath.ac.uk
nagman:receiving results from dictum.maths.bath.ac.uk
```

The method first discretizes the PDE using the routine `d03eef` which creates the seven-diagonal system of finite difference equations. These are then passed onto the routine `d03edf` for solution by a multigrid technique.

```
(2)
[ifail: 0, us: Matrix(DoubleFloat), rhs: Matrix(DoubleFloat),
 u: Matrix(DoubleFloat), numit: 3,

  method:
    [nameOfRoutine: "d03eefAnnaType", allMeasures: List(String),
     bestMeasure: 0.5]
  ,
 ub: Matrix(DoubleFloat), a: Matrix(DoubleFloat)]
                                                          Type: Result
```

A.5 Optimization

Problem: Minimize $x_1x_4(x_1+x_2+x_3)+x_3$

$$
\begin{aligned}
1 < x_1 < 5 \\
1 < x_2 < 5 \\
1 < x_3 < 5 \\
1 < x_4 < 5 \\
-\infty < x_1+x_2+x_3+x_4 < 20 \\
-\infty < x_1^2+x_2^2+x_3^2+x_4^2 < 40 \\
-25 < x_1x_2x_3x_4 < \infty
\end{aligned}
$$

with constraints:

and initial guess: $[1.0, 5.0, 5.0, 1.0]$

```
Axiom Computer Algebra System (Release 2.1)
Solaris 2.5 for Sun SPARC [SunPro cc]
------------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave AXIOM and return to shell.
------------------------------------------------------------------------------

Value = "ANNA loaded successfully"
(1) ->
(1) -> showScalarValues true;

                                                                   Type: Boolean
(2) -> optimize(X[1]*X[4]*(X[1]+X[2]+X[3])+X[3], [1.0, 5.0, 5.0, 1.0],
[1.0, 1.0, 1.0, 1.0, -1.E25, -1.E25, 25.0], [X[1]+X[2]+X[3]+X[4] ,
X[1]**2+X[2]**2+X[3]**2+X[4]**2 , X[1]*X[2]*X[3]*X[4] ], [5.0, 5.0, 5.0,
5.0, 20.0, 40.0, 1.E25])
nagman:acknowledging request for e04ucf
nagman:connection successful to dictum.maths.bath.ac.uk
nagman:receiving results from dictum.maths.bath.ac.uk

   (2)
   [ifail: 0, c: Matrix(DoubleFloat), objf: 17.0140172891347,
    objgrd: Matrix(DoubleFloat), attributes: List(String), iter: 5,
    clamda: Matrix(DoubleFloat), x: Matrix(DoubleFloat),
    istate: Matrix(Integer), r: Matrix(DoubleFloat),
    cjac: Matrix(DoubleFloat),

    method:
       [nameOfRoutine: "e04ucfAnnaType", allMeasures: List(String),
```

```
        bestMeasure: 0.6]
     ]
                                                                Type: Result
(3) -> qelt(%.method,allMeasures)

   (3)
   [
    "e04mbfmeasure: 0.0 - e04mbf is for a linear objective function and
     constraints only.",
    "e04nafmeasure: 0.0 - e04naf is for a quadratic function with linear
     constraints only.",
    "e04ucfmeasure: 0.6 - e04ucf is recommended",
    "e04dgf is no better than other routines",
    "e04gcfmeasure: 0.0 - e04gcf is unsuitable for constrained problems. ",
    "e04jaf is no better than other routines",
    "e04fdfmeasure: 0.0 - e04fdf is unsuitable for constrained problems. "]
                                                          Type: List String
(4) -> %%(2).attributes

   (4)
   ["The object function is non-linear",
    "There are simple bounds on the variables",
    "There are 1 linear and 2 non-linear constraints"]
                                                          Type: List String
```

The count of linear and non-linear constraints excludes the simple bounds on the variables.

```
(5) -> %%(2).x

   (5)  [1.0  4.7429996428483  3.82114997689538  1.37940829417858]
                                                  Type: Matrix DoubleFloat
(6) -> %%(2).objf

   (6)  17.0140172891347
                                                          Type: DoubleFloat
```

Problem: Calculate a least-squares minimum of:

$$
\begin{aligned}
&(x_3 - 15x_2)^{-1} + x_1 - 0.14 \\
&2(2x_3 - 14x_2)^{-1} + x_1 - 0.18 \\
&3(3x_3 - 13x_2)^{-1} + x_1 - 0.22 \\
&4(4x_3 - 12x_2)^{-1} + x_1 - 0.25 \\
&5(5x_3 - 11x_2)^{-1} + x_1 - 0.29 \\
&6(6x_3 - 10x_2)^{-1} + x_1 - 0.32 \\
&7(7x_3 - 9x_2)^{-1} + x_1 - 0.35 \\
&8(8x_3 - 8x_2)^{-1} + x_1 - 0.39 \\
&9(9x_3 - 7x_2)^{-1} + x_1 - 0.37 \\
&10(10x_3 - 6x_2)^{-1} + x_1 - 0.58 \\
&11(11x_3 - 5x_2)^{-1} + x_1 - 0.73 \\
&12(12x_3 - 4x_2)^{-1} + x_1 - 0.96 \\
&13(13x_3 - 3x_2)^{-1} + x_1 - 1.34 \\
&14(15x_3 - 2x_2)^{-1} + x_1 - 2.1 \\
&15(15x_3 - x_2)^{-1} + x_1 - 4.39
\end{aligned}
$$

```
Axiom Computer Algebra System (Release 2.1)
Solaris 2.5 for Sun SPARC [SunPro cc]
------------------------------------------------------------------------------
   Issue )copyright to view copyright notices.
   Issue )summary for a summary of useful system commands.
   Issue )quit to leave AXIOM and return to shell.
------------------------------------------------------------------------------

Value = "ANNA loaded successfully"
(1) ->
(1) -> showScalarValues true;

                                                                   Type: Boolean
(2) -> optimize( [(X[3]+15*X[2])**(-1)+X[1]-0.14 , 2*(2*X[3]+14*X[2])**(-1)
+X[1]-0.18 , 3*(3*X[3]+13*X[2])**(-1)+X[1]-0.22 , 4*(4*X[3]+12*X[2])**(-1)
+X[1]-0.25 , 5*(5*X[3]+11*X[2])**(-1)+X[1]-0.29 , 6*(6*X[3]+10*X[2])**(-1)
+X[1]-0.32 , 7*(7*X[3]+9*X[2])**(-1)+X[1]-0.35 , 8*(8*X[3]+8*X[2])**(-1)
+X[1]-0.39 , 9*(7*X[3]+7*X[2])**(-1)+X[1]-0.37 , 10*(6*X[3]+6*X[2])**(-1)
+X[1]-0.58 , 11*(5*X[3]+5*X[2])**(-1)+X[1]-0.73 , 12*(4*X[3]+4*X[2])**(-1)
+X[1]-0.96 , 13*(3*X[3]+3*X[2])**(-1)+X[1]-1.34 , 14*(2*X[3]+2*X[2])**(-1)
+X[1]-2.1 , 15*(X[3]+X[2])**(-1)+X[1]-4.39 ], [0.5, 1.0, 1.5])
nagman:acknowledging request for e04gcf
nagman:connection successful to dictum.maths.bath.ac.uk
```

```
nagman:receiving results from dictum.maths.bath.ac.uk


   (2)
   [ifail: 0, w: Matrix(DoubleFloat),

     method:
       [nameOfRoutine: "e04gcfAnnaType", allMeasures: List(String),
        bestMeasure: 0.7214220332 0194042508]
     ,
    attributes: List(String), x: Matrix(DoubleFloat),
    objf: 0.00821487730657901]
                                                          Type: Result
(3) -> qelt(%.method,allMeasures)

   (3)
   ["e04gcfmeasure: 0.721 - e04gcf is recommended.",
    "e04fdf is no better than other routines"]
                                                    Type: List String
(4) -> %%(2).attributes

   (4)  ["The object functions are non-linear"]
                                                    Type: List String
(5) -> %%(2).x

   (5)  [0.0824105598097718  1.13303609205156  2.34369517871324]
                                                 Type: Matrix DoubleFloat
(6) -> %%(2).objf

   (6)  0.00821487730657901
                                                    Type: DoubleFloat
```

Appendix B

Code Production and Testing Procedures

B.1 ANNA Categories Domains and Packages

B.1.1 Categories

- `NumericalIntegrationCategory`
- `OrdinaryDifferentialEquationsSolverCategory`
- `PartialDifferentialEquationsSolverCategory`
- `NumericalOptimizationCategory`

B.1.2 Method Domains

Integration

- `d01ajfAnnaType`
- `d01akfAnnaType`
- `d01alfAnnaType`
- `d01amfAnnaType`
- `d01anfAnnaType`
- `d01apfAnnaType`
- `d01aqfAnnaType`
- `d01asfAnnaType`
- `d01fcfAnnaType`
- `d01gbfAnnaType`
- `d01TransformFunctionType`

Ordinary Differential Equations

- d02bbfAnnaType
- d02bhfAnnaType
- d02cjfAnnaType
- d02ejfAnnaType

Partial Differential Equations

- d03eefAnnaType

Optimization

- e04dgfAnnaType
- e04fdfAnnaType
- e04gcfAnnaType
- e04jafAnnaType
- e04mbfAnnaType
- e04nafAnnaType
- e04ucfAnnaType

B.1.3 Packages

Top Level Packages

- AnnaNumericalIntegrationPackage
- AnnaOrdinaryDifferentialEquationPackage
- AnnaPartialDifferentialEquationPackage
- AnnaNumericalOptimizationPackage

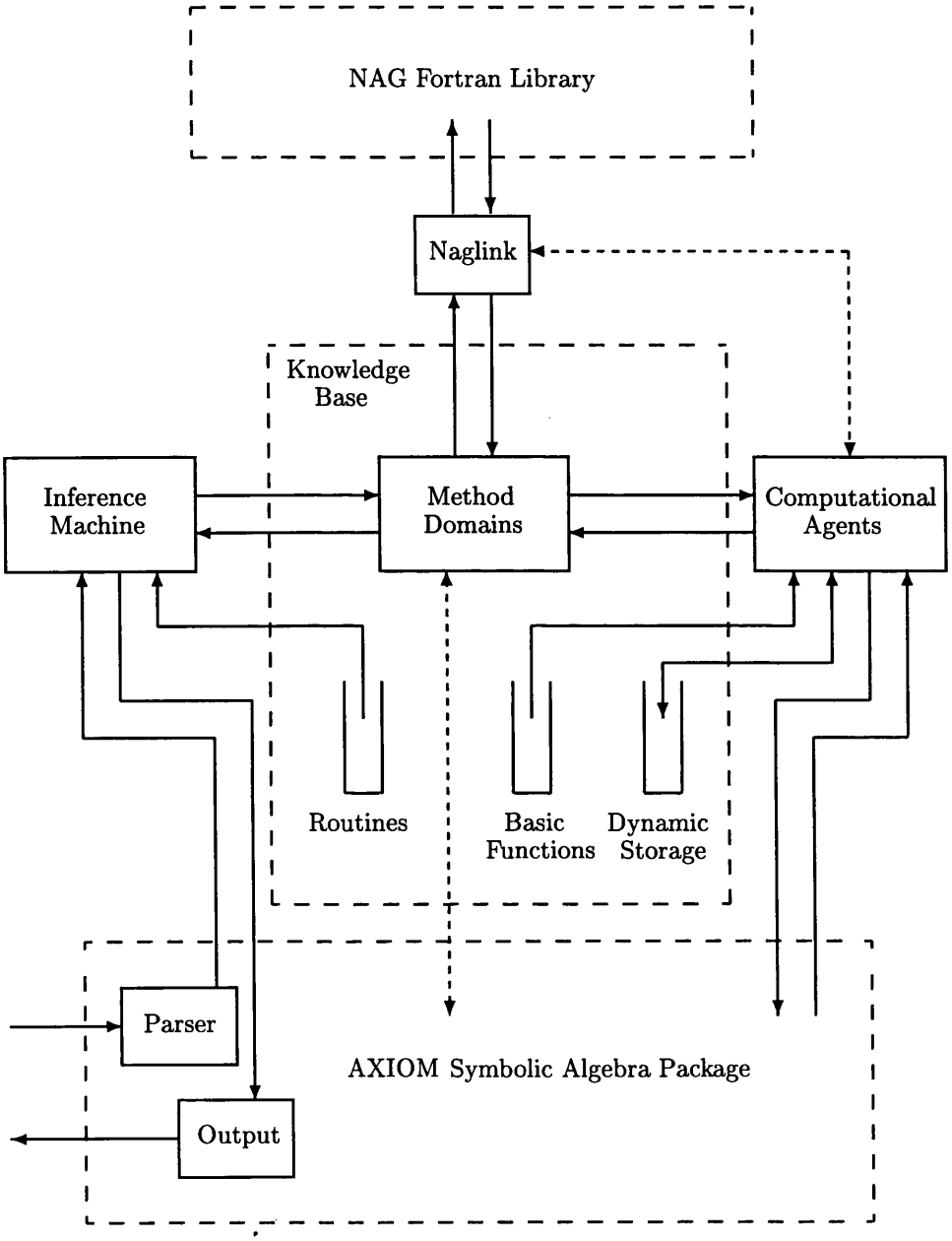
Computational Agent Packages

- d01AgentsPackage
- d02AgentsPackage
- d03AgentsPackage
- e04AgentsPackage
- ExpertSystemContinuityPackage
- ExpertSystemContinuityPackage1
- d01WeightsPackage

B.1.4 Miscellaneous Domains

- `RoutinesTable`
- `BasicFunctions`
- `AttributeButtons`
- `IntegrationFunctionsTable`
- `ODEIntensityFunctionsTable`
- `ExpertSystemToolsPackage`
- `ExpertSystemToolsPackage1`
- `ExpertSystemToolsPackage2`

B.2 Structural Design



B.3 Testing and Evaluation

B.3.1 Integration

Integration Problem	Recommended Routine(s)
$\int_0^\infty \frac{e^{-3x^2} + e^{-x^3}}{\sqrt{x}}\, dx$	d01APF & D01AJF[1]
$\int_0^\infty \frac{e^y}{\sqrt{y}}\, dy$	D01APF & D01AJF[1]
$\int_0^2 \frac{e^{-3x^2} + e^{-x^3}}{(x^2-2)}\, dx$	D01AQF
$\int_0^2 \frac{1}{(x^6-1)}\, dx$	D01AQF
$\int_{-\pi}^{\pi} \cos t^2 + \sin t + \cos\sin t^3\, dt$	D01AKF
$\int_{-1}^{1} x \prod_{i=-4}^{4} \left(x - \frac{i}{10}\right) dx$	D01AKF
$\int_0^1 \cos 20x\, (\sin x^2 + \cos x^2)\, dx$	D01ANF
$\int_0^2 \frac{\log x\, \log(2-x)}{\sqrt{x}\, \sqrt[3]{(2-x)^2}}\, dx$	D01APF
$\frac{1}{\sqrt{2\pi}} \int_0^\infty e^{-z^2/2}\, dz$	D01AMF

[1]The ANNA routine D01Transform splits the function, transforms one part, and uses the routines D01APF and D01AJF on the two parts

Integration Problem	Recommended Routine(s)
$\int_{-\infty}^{\infty} \frac{e^{-\omega}}{(\omega-5)\ (\omega-\frac{1}{2})}\ d\omega$	D01AQF & D01AQF[2]
$\int_{-1}^{1} \log u^2\ du$	D01ALF
$\int_0^1 \int_0^1 \int_0^1 \int_0^1 4x_1x_3^2 e^{\frac{2x_1x_3}{(1+x_2+x_4)^2}}\ dx_4\ dx_3\ dx_2\ dx_1$	D01FCF

[2]The ANNA routine D01Transform splits the function and uses the routine D01AQF on the two separate parts

B.3.2 Ordinary Differential Equations

Non-stiff Equations[3]

Ref	System of Equations	Initial Conditions	Routine
N[4]	$y_1' = \tan y_3$ $y_2' = -0.032\frac{\tan y_3}{y_2} - 0.02\frac{y_2}{\cos y_3}$ $y_3' = -\frac{0.032}{y_2^2}$	$y_1(0) = 0.5$ $y_2(0) = 0.5$ $y_3(0) = 0.2\pi$	D02BBF
A1	$y' = -y$	$y(0) = 1$	D02BBF
A2	$y' = -y^3/2$	$y(0) = 1$	D02BBF
A3	$y' = y \cos x$	$y(0) = 1$	D02BBF
A4	$y' = \frac{y}{4}\left(1 - \frac{y}{20}\right)$	$y(0) = 1$	D02BBF
A5	$y' = \frac{y-x}{y+x}$	$y(0) = 4$	D02BBF

[3]Most of the test examples are from [Hull *et al.*, 1972]
[4]This example is used in [NAG, 1996]

Ref	System of Equations	Initial Conditions	Routine
B1	$y_1' = 2(y_1 - y_1y_2)$ $y_2' = -(y_2 - y_1y_2)$	$y_1(0) = 1$ $y_2(0) = 3$	D02BBF
B2	$y_1' = -y_1 + y_2$ $y_2' = y_1 - 2y_2 + y_3$ $y_3' = y_2 - y_3$	$y_1(0) = 2$ $y_2(0) = 0$ $y_3(0) = 1$	D02BBF
B3	$y_1' = -y_1$ $y_2' = y_1 - y_2^2$ $y_3' = y_2^2$	$y_1(0) = 1$ $y_2(0) = 0$ $y_3(0) = 0$	D02BBF
B4	$y_1' = -y_2 - y_1y_3\sqrt{y_1^2 + y_2^2}$ $y_2' = y_1 - y_2y_3\sqrt{y_1^2 + y_2^2}$ $y_3' = y_1\sqrt{y_1^2 + y_2^2}$	$y_1(0) = 3$ $y_2(0) = 0$ $y_3(0) = 0$	D02CJF
B5	$y_1' = y_2y_3$ $y_2' = -y_1y_3$ $y_3' = -.51y_1y_2$	$y_1(0) = 0$ $y_2(0) = 1$ $y_3(0) = 1$	D02BBF

Ref	System of Equations	Initial Conditions	Routine
C1	$\begin{bmatrix} y_1' \\ y_2' \\ . \\ . \\ y_{10}' \end{bmatrix} = \begin{bmatrix} -1 & & & & \\ 1 & -1 & & 0 & \\ & 1 & . & & \\ & & . & . & \\ & 0 & & . & -1 \\ & & & 1 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_{10} \end{bmatrix}$	$y(0) = \begin{bmatrix} 1 \\ 0 \\ . \\ . \\ 0 \end{bmatrix}$	D02BBF
C2	$\begin{bmatrix} y_1' \\ y_2' \\ . \\ . \\ y_{10}' \end{bmatrix} = \begin{bmatrix} -1 & & & & \\ 1 & -2 & & 0 & \\ & 2 & -3 & & \\ & & . & . & \\ & 0 & & . & -9 \\ & & & 9 & 0 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_{10} \end{bmatrix}$	$y(0) = \begin{bmatrix} 1 \\ 0 \\ . \\ . \\ 0 \end{bmatrix}$	D02BBF
C3	$\begin{bmatrix} y_1' \\ y_2' \\ . \\ . \\ y_{10}' \end{bmatrix} = \begin{bmatrix} -2 & 1 & & & \\ 1 & -2 & 1 & 0 & \\ & 1 & . & & \\ & & . & . & \\ & 0 & & . & -2 & 1 \\ & & & & 1 & -2 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ . \\ . \\ y_{10} \end{bmatrix}$	$y(0) = \begin{bmatrix} 1 \\ 0 \\ . \\ . \\ 0 \end{bmatrix}$	D02BBF
C4	As C3 except with 51 equations		D02CJF

Ref	System of Equations	Initial Conditions	Routine
D1	$y_1' = y_3$ $y_2' = y_4$ $y_3' = -y_1(y_1^2+y_2^2)^{3/2}$ $y_4' = -y_2(y_1^2+y_2^2)^{3/2}$	$y_1(0) = 1-\epsilon$ $y_2(0) = 0$ $y_3(0) = 0$ $y_4(0) = \sqrt{\frac{1+\epsilon}{1-\epsilon}}$ $\epsilon = 0.1$	D02BBF
D2	As above with $\epsilon = 0.3$		D02BBF
D3	As above with $\epsilon = 0.5$		D02BBF
D4	As above with $\epsilon = 0.7$		D02BBF
D5	As above with $\epsilon = 0.9$		D02BBF

Ref	System of Equations	Initial Conditions	Routine
E1	$y_1' = y_2$ $y_2' = -\left(\frac{y_2}{x+1} + \left(1 - \frac{0.25}{(x+1)^2}\right) y_1\right)$	$y_1(0) = J_{1/2}(1)$[5] $y_2(0) = J'_{1/2}(1)$[6]	D02BBF
E2	$y_1' = y_2$ $y_2' = (1 - y_1^2)y_2 - y_1$	$y_1(0) = 2$ $y_2(0) = 0$	D02CJF
E3	$y_1' = y_2$ $y_2' = \frac{y_1^3}{6} - y_1 + 2\sin(2.78535x)$	$y_1(0) = 0$ $y_2(0) = 0$	D02CJF
E4	$y_1' = y_2$ $y_2' = 0.032 - 0.4y_2^2$	$y_1(0) = 30$ $y_2(0) = 0$	D02CJF
E5	$y_1' = y_2$ $y_2' = \frac{\sqrt{1+y_2^2}}{25-x}$	$y_1(0) = 0$ $y_2(0) = 0$	D02CJF

[5] 0.6713967071418030
[6] 0.09540051444747446

Stiff and Mildly-Stiff Equations[7]

Ref	System of Equations	Initial Conditions	Routine
N[8]	$y_1' = -\frac{1}{25}y_1 + 10^4 y_2 y_3$ $y_2' = \frac{1}{25}y_1 - 10^4 y_2 y_3 - 3 \times 10^7 y_2^2$ $y_3' = 3 \times 10^7 y_2^2$	$y_1(0) = 1$ $y_2(0) = 0$ $y_3(0) = 0$	D02EJF
A1	$y_1' = -0.5y_1$ $y_2' = -y_2$ $y_3' = -100y_3$ $y_4' = -90y_4$	$y_1(0) = 1$ $y_2(0) = 1$ $y_3(0) = 1$ $y_4(0) = 1$	D02CJF[9]
A2	$y_1' = -1800y_1 + 900y_2$ $y_i' = -y_{i-1} - 2y_i + y_{i+1}$ $y_9' = -1000y_8 - 2000y_9 + 1000$ $i = 2, ..., 8$	$y_1(0) = 0$ $y_i(0) = 0$ $y_8(0) = 0$	D02EJF
A3	$y_1' = -10^4 y_1 + 100y_2 - 10y_3 + y_4$ $y_2' = -10^3 y_2 + 10y_3 - 10y_4$ $y_3' = -y_3 + 10y_4$ $y_4' = -0.1y_4$	$y_1(0) = 1$ $y_2(0) = 1$ $y_3(0) = 1$ $y_4(0) = 1$	D02EJF
A4	$y_i' = -i^5 y_i$ $i = 1, 2, ..., 10$	$y_i(0) = 1$	D02EJF

[7]Most of the test examples are from [Enright *et al.*, 1975]
[8]This example is used in [NAG, 1996]
[9]The system of equations is only mildly stiff

Ref	System of Equations	Initial Conditions	Routine
B1	$y_1' = -y_1 + y_2$ $y_2' = -100y_1 - y_2$ $y_3' = -100y_3 + y_4$ $y_4' = -10000y_3 - 100y_4$	$y_1(0) = 1$ $y_2(0) = 0$ $y_3(0) = 1$ $y_4(0) = 0$	D02BBF[10]
B2	$y_1' = -10y_1 + \alpha y_2$ $y_2' = -\alpha y_1 - 10y_2$ $y_3' = -4y_3$ $y_4' = -y_4$ $y_5' = -0.5y_5$ $y_6' = -0.1y_6$ $\alpha = 3$	$y_1(0) = 1$ $y_2(0) = 1$ $y_3(0) = 1$ $y_4(0) = 1$ $y_5(0) = 1$ $y_6(0) = 1$	D02BBF[10]
B3	As in B2 with $\alpha = 8$		D02BBF[10]
B4	As in B2 with $\alpha = 25$		D02BBF[10]
B5	As in B2 with $\alpha = 100$		D02BBF[10]

[10]The system of equations is only very mildly stiff

Ref	System of Equations	Initial Conditions	Routine
C1	$y_1' = -y_1 + y_2^2 + y_3^2 + y_4^2$ $y_2' = -10y_2 + 10(y_3^2 + y_4^2)$ $y_3' = -40y_3 + 40y_4^2$ $y_4' = -100y_4 + 2$	$y_1(0) = 1$ $y_2(0) = 1$ $y_3(0) = 1$ $y_4(0) = 1$	D02CJF[11]
C2	$y_1' = -y_1 + 2$ $y_2' = -10y_2 + \beta y_1^2$ $y_3' = -40y_3 + 4\beta(y_1^2 + y_2^2)$ $y_4' = -100y_4 + 10\beta(y_1^2 + y_2^2 + y_3^2)$ $\beta = 0.1$	$y_1(0) = 1$ $y_2(0) = 1$ $y_3(0) = 1$ $y_4(0) = 1$	D02CJF[11]
C3	As in C2 with $\beta = 1$		D02CJF[11]
C4	As in C2 with $\beta = 10$		D02CJF[11]
C5	As in C2 with $\beta = 20$		D02CJF[11]

[11]The system of equations is only mildly stiff

Ref	System of Equations	Initial Conditions	Routine
D1	$y_1' = 0.2(y_2 - y_1)$ $y_2' = 10y_1 - (60 - 0.125y_3)y_2 + 0.125y_3$ $y_3' = 1$	$y_1(0) = 0$ $y_2(0) = 0$ $y_3(0) = 0$	D02CJF[12]
D2	$y_1' = -0.4y_1 + 0.01y_2y_3$ $y_2' = 400y_1 - 100y_2y_3 - 3000y_2^2$ $y_3' = 30y_2^2$	$y_1(0) = 1$ $y_2(0) = 0$ $y_3(0) = 0$	D02EJF
D3	$y_1' = y_3 - 100y_1y_2$ $y_2' = y_3 + 2y_4 - 100y_1y_2 - 2 \times 10^2y_2^2$ $y_3' = -y_3 + 100y_1y_2$ $y_4' = -y_4 + 10^4y_2^2$	$y_1(0) = 1$ $y_2(0) = 1$ $y_3(0) = 0$ $y_4(0) = 0$	D02CJF[12]
D4	$y_1' = -0.013y_1 - 1000y_1y_2$ $y_2' = -2500y_2y_3$ $y_3' = -0.013y_1 - 1000y_1y_3 - 2500y_2y_3$	$y_1(0) = 1$ $y_2(0) = 1$ $y_3(0) = 0$	D02BBF[13]
D5	$y_1' = 0.01 - (1 + (y_1 + 1000)(y_1 + 1))(0.01 + y_1 + y_2)$ $y_2' = 0.01 - (1 + y_2^2)(0.01 + y_1 + y_2)$	$y_1(0) = 0$ $y_2(0) = 0$	D02EJF
D6	$y_1' = -y_1 + 10^8y_3(1 - y_1)$ $y_2' = -10y_2 + 3 \times 10^7y_3(1 - y_2)$ $y_3' = -y_1' - y_2'$	$y_1(0) = 1$ $y_2(0) = 0$ $y_3(0) = 0$	D02EJF

[12]The system of equations is only mildly stiff

[13]ANNA identifies that the system of equations is very stiff but still incorrectly chooses the wrong routine

Ref	System of Equations	Initial Conditions	Routine
E1	$y_1' = y_2$ $y_2' = y_3$ $y_3' = y_4$ $y_4' = (y_1^2 - \sin(y_1) - \Gamma^4)y_1 + \left(\frac{y_2 y_3}{y_1^2+1} - 4\Gamma^3\right) y_2 + (1 - 6\Gamma^2)y_3 + (10e^{-y_4^2} - 4\Gamma)y_4 + 1$ $\Gamma = 100$	$y_1(0) = 0$ $y_2(0) = 0$ $y_3(0) = 0$ $y_4(0) = 0$	D02BBF[14]
E2	$y_1' = y_2$ $y_2' = 5(1 - y_1^2)y_2 - y_1$	$y_1(0) = 2$ $y_2(0) = 0$	D02CJF[14]
E3	$y_1' = -(55 + y_3)y_1 + 65y_2$ $y_2' = 0.785(y_1 - y_2)$ $y_3' = 0.01y_1$	$y_1(0) = 1$ $y_2(0) = 1$ $y_3(0) = 0$	D02BBF[15]
E5	$y_1' = -0.789 \times 10^{-10} y_1 - 1.1 \times 10^7 y_1 y_3$ $y_2' = 7.89 \times 10^{-10} y_1 - 1.13 \times 10^9 y_2 y_3$ $y_3' = 7.89 \times 10^{-10} y_1 - 1.1 \times 10^7 y_1 y_3 + 1.13 \times 10^3 y_4 - 1.13 \times 10^9 y_2 y_3$ $y_4' = 1.1 \times 10^7 y_1 y_3 - 1.13 \times 10^3 y_4$	$y_1(0) = 0.00176$ $y_2(0) = 0$ $y_3(0) = 0$ $y_4(0) = 0$	D02EJF

[14]The system of equations is only mildly stiff

[15]The stiffness of the system of equations increases over the integration period and is thus not correctly identified by ANNA's detection algorithm. It thus chooses a non-optimal routine

B.3.3 Optimization

Minimization of a Single Multivariate Function

Function	Constraints	Routine
$e_1^x(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1)$		E04DGF
$(x_1 + 10x_2)^2 + 5(x_3 - x_4)^2$ $+(x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$	$1 < x_1 < 3$ $-2 < x_2 < 0$ $-\infty < x_3 < \infty$ $1 < x_4 < 3$	E04JAF
$x_1x_4(x_1 + x_2 + x_3) + x_3$	$1 < x_1 < 5$ $1 < x_2 < 5$ $1 < x_3 < 5$ $1 < x_4 < 5$ $-\infty < x_1 + x_2 + x_3 + x_4 < 20$ $-\infty < x_1^2 + x_2^2 + x_3^2 + x_4^2 < 40$ $-25 < x_1x_2x_3x_4 < \infty$	E04UCF

Function	Constraints	Routine
$-0.2(0.1x_1 + x_2 + x_3 + x_4 + x_5 - 0.2(x_6 + x_7))$	$-0.01 < x_1 < 0.01$ $-0.1 < x_2 < 0.15$ $-0.01 < x_3 < 0.03$ $-0.04 < x_4 < 0.02$ $-0.1 < x_5 < 0.05$ $-0.01 < x_6 < \infty$ $-0.01 < x_7 < \infty$ $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 = -0.13$ $-\infty < 0.15x_1 + 0.04x_2 + 0.02x_3 + 0.04x_4 + 0.02x_5 + 0.01x_6 + 0.03x_7 < -0.0049$ $-\infty < 0.03x_1 + 0.05x_2 + 0.08x_3 + 0.02x_4 + 0.06x_5 + 0.01x_6 < -0.0064$ $-\infty < 0.02x_1 + 0.04x_2 + 0.01x_3 + 0.02x_4 + 0.02x_5 < -0.0037$ $-\infty < 0.02x_1 + 0.03x_2 + 0.01x_5 < -0.0012$ $-0.0992 < 0.7x_1 + 0.75x_2 + 0.8x_3 + 0.75x_4 + 0.8x_5 + 0.97x_6 < \infty$ $-0.003 < 0.02x_1 + 0.06x_2 + 0.08x_3 + 0.12x_4 + 0.02x_5 + 0.01x_6 + 0.97x_7 < 0.002$	E04MBF

Function	Constraints	Routine
$(x_1 - 0.02)x_1 + (x_2 - 0.2)x_2$ $+(x_3 - 0.2)x_3$ $+(x_4 + 2x_3 - 0.2)x_4$ $+(x_5 - 0.2)x_5$ $+(0.04 - x_6)x_6$ $+(0.04 - 2x_6 - x_7)x_7$	$-0.01 < x_1 < 0.01$ $-0.1 < x_2 < 0.15$ $-0.01 < x_3 < 0.03$ $-0.04 < x_4 < 0.02$ $-0.1 < x_5 < 0.05$ $-0.01 < x_6 < \infty$ $-0.01 < x_7 < \infty$ $x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 = -0.13$ $-\infty < 0.15x_1 + 0.04x_2 + 0.02x_3 + 0.04x_4 + 0.02x_5 + 0.01x_6 + 0.03x_7 < -0.0049$ $-\infty < 0.03x_1 + 0.05x_2 + 0.08x_3 + 0.02x_4 + 0.06x_5 + 0.01x_6 < -0.0064$ $-\infty < 0.02x_1 + 0.04x_2 + 0.01x_3 + 0.02x_4 + 0.02x_5 < -0.0037$ $-\infty < 0.02x_1 + 0.03x_2 + 0.01x_5 < -0.0012$ $-0.0992 < 0.7x_1 + 0.75x_2 + 0.8x_3 + 0.75x_4 + 0.8x_5 + 0.97x_6 < \infty$ $-0.003 < 0.02x_1 + 0.06x_2 + 0.08x_3 + 0.12x_4 + 0.02x_5 + 0.01x_6 + 0.97x_7 < 0.002$	E04NAF

Least-Squares Problem

Functions	Routine
$(x_3 - 15x_2)^{-1} + x_1 - 0.14$ $2(2x_3 - 14x_2)^{-1} + x_1 - 0.18$ $3(3x_3 - 13x_2)^{-1} + x_1 - 0.22$ $4(4x_3 - 12x_2)^{-1} + x_1 - 0.25$ $5(5x_3 - 11x_2)^{-1} + x_1 - 0.29$ $6(6x_3 - 10x_2)^{-1} + x_1 - 0.32$ $7(7x_3 - 9x_2)^{-1} + x_1 - 0.35$ $8(8x_3 - 8x_2)^{-1} + x_1 - 0.39$ $9(9x_3 - 7x_2)^{-1} + x_1 - 0.37$ $10(10x_3 - 6x_2)^{-1} + x_1 - 0.58$ $11(11x_3 - 5x_2)^{-1} + x_1 - 0.73$ $12(12x_3 - 4x_2)^{-1} + x_1 - 0.96$ $13(13x_3 - 3x_2)^{-1} + x_1 - 1.34$ $14(15x_3 - 2x_2)^{-1} + x_1 - 2.1$ $15(15x_3 - x_2)^{-1} + x_1 - 4.39$	E04GCF